AD-A089 542 STANFORD UNIV CA DEPT OF OPERATIONS RESEARCH F/G 12/2
THE STAIRCASE AND RELATED STRUCTURES IN INTEGER PROGRAMMING.(U)
JUN 80 L J POLLENZ N00014-76-C-0418
UNCLASSIFIED TR-94 NL

1 OF 2
AD A089542

THE STAIRCASE AND RELATED STRUCTURE IN INTEGER PROGRAMMING

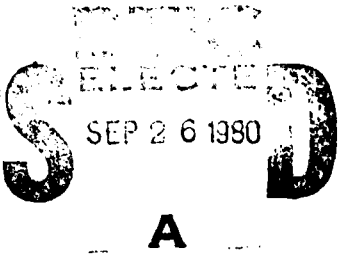
BY

LYNNE J. POLLENZ

TECHNICAL REPORT NO. 94
JUNE 1980

PREPARED UNDER CONTRACT
N00014-76-C-0418 (NR-047-061)
FOR THE OFFICE OF NAVAL RESEARCH

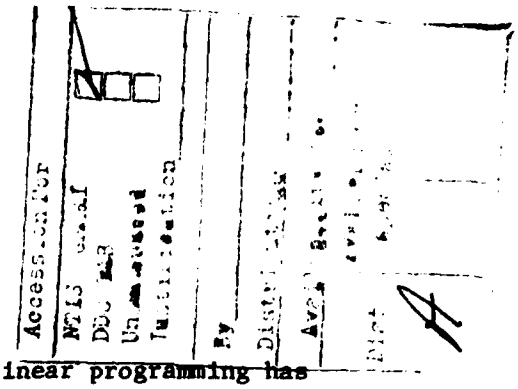
Frederick S. Hillier, Project Director



DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION



1. Problem Definition

With the advent of the computer age, linear programming has become a major tool for practical problem solving. Integer programming applications are also widespread, but unfortunately no integer programming solution method has been as successful as the simplex method for linear programming. However, it is often possible to improve considerably on computation time by exploiting the special structure of a problem. One common special structure, the staircase structure, occurs frequently in multitime period models.

The staircase integer programming problem can be written in the following form:

$$
\begin{array}{lll}
 & \text{maximize} & cx \\
(P) & \text{subject to} & Ax \leq b \\
 & & x \geq 0 \\
 & & x \text{ integer}
\end{array}
$$

where the cost coefficients form the n-vector c, the right-hand side b is an m-vector, and x is an n-vector of nonnegative integral variables. The $m \times n$ constraint matrix A has the special structure depicted in Figure 1.

Such a matrix A, with all of its nonzero elements found in blocks centered roughly on and just below the main diagonal, is called a staircase matrix because of its resemblance to a set of steps. The submatrices A_t, $t = 1, \ldots, T$, are called diagonal blocks and are of

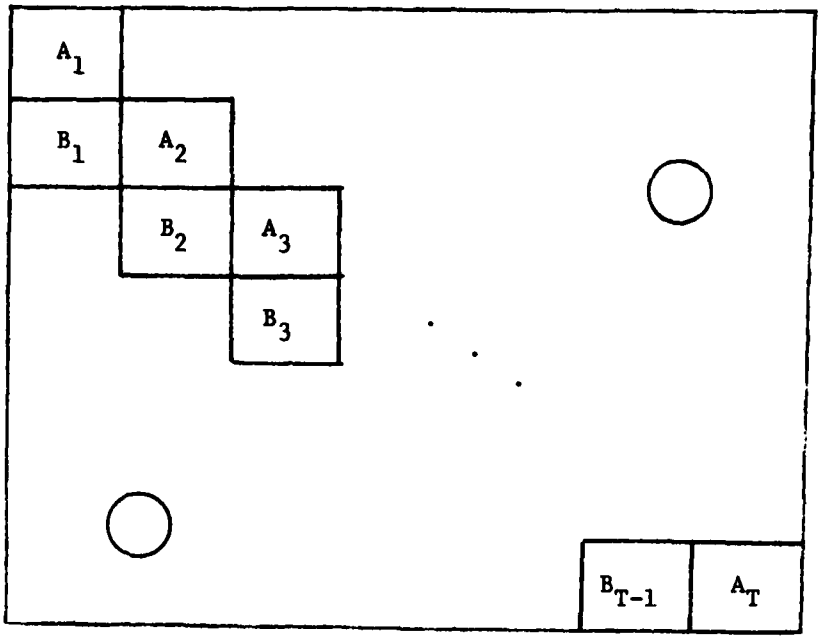


Figure 1: Staircase Matrix

dimensions $m_t \times n_t$, where $\sum_{t=1}^{T} m_t = m$ and $\sum_{t=1}^{T} n_t = n$. The off-diagonal blocks are of size $m_{t-1} \times n_t$, and are represented by submatrices B_t, $t = 1, \ldots, T - 1$, in Figure 1. For any nonzero column of an offdiagonal block, the associated column of A is called a linking column, with the corresponding linking variable being the appropriate component of the vector x. On the other hand, an all zero column in B_t is associated with a variable that is said to be local to period t (since it has no effect on period $t + 1$ through the matrix B_t).

If the constraint matrix A is not required to have any particular structure, problem (P) is called an integer linear programming problem (henceforth abbreviated ILP). Omission of all the integrality constraints on the variables results in a linear programming problem (abbreviated LP) which is called the LP relaxation of problem (P). If

only a subset of the variables are constrained to be integral, a mixed integer linear program (MILP) is obtained. Although the methods developed in the next chapter assume a pure integer programming formulation, extensions to the MILP case will be examined in Chapter 5.

Problem (P) can alternatively be formulated as a linear optimal control problem:

$$\text{maximize} \quad c_t x_t$$

$$\text{(CP)} \qquad \text{subject to} \quad A_t x_t = b_t - B_{t-1} x_{t-1} - u_t , \quad t = 1, \ldots, T$$

$$x_t, u_t \geq 0, \; x_o = 0$$

$$x_t \text{ integral}$$

The optimal control problem is to choose a sequence of admissable controls u_t and system states x_t maximizing the objective function. Unfortunately, control theory methods have not been developed to deal with the discrete nature of the integrality restrictions (see Luenberger [1979]).

The above formulation can be generalized to an r^{th} order control problem, in which the equality constraints of (CP) would involve $r + 1$ consecutive system states. These constraints are given by:

$$A_t x_t = b_t - \sum_{i=1}^{\min\{r,t-1\}} B_{t,t-i} x_{t-i} - u_t , \quad t = 1, \ldots, T$$

Analogously, the period t constraints of an r^{th} order staircase matrix link activities from periods k, $t - r \leq k \leq t$. Carrying this generalization to the extreme results in the lower block triangular form, an example of which is shown in Figure 2. The solution techniques developed in succeeding chapters can be extended to the r^{th} order case, as we will discuss in Chapter 5.

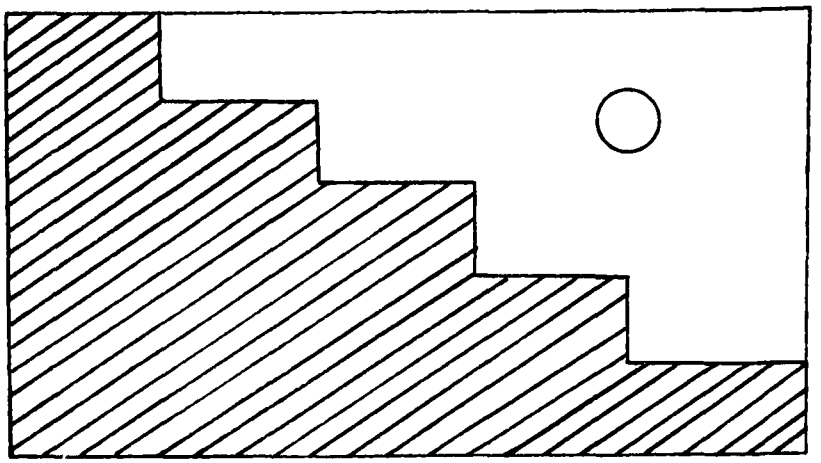


Figure 2: Block Triangular Matrix

Staircase structured constraint matrices arise in a wide variety of practical applications. Some notable examples which have been examined in the literature are: multiplant production allocation problems (Driebeek [1969]), optimal design of multistage structures (Ho [1975]), multisector economic planning models (Manne [1970]), and multitime period production and inventory problems (Glassey [1971], Lasdon [1970]). Of these applications, the latter are perhaps the most representative. For this reason, most of the terminology used in this paper will draw heavily upon the jargon associated with multitime period models.

2. Integer Programming Solution Techniques

One way in which to solve the staircase integer programming problem (P) is to ignore the special structure of the constraint matrix and apply one of the standard integer programming algorithms. These algorithms fall into three major categories: cutting plane methods, group theoretic methods, and implicit enumeration methods. The discussion of these techniques here will necessarily be brief; a more comprehensive survey is given by Geoffrion and Marsten [1972], and a fuller treatment of the theory of integer programming can be found in Garfinkel and Nemhauser [1972].

The cutting plane approach to integer programming utilizes the simplex method to solve a succession of linear programming problems for which the optimal solutions converge to an optimal integral solution to the original problem (see Gomory [1963]). Initially the LP relaxation of the ILP is solved, and if the optimal solution x^o is integral, the algorithm terminates. Otherwise, an extra linear constraint or "cut" is added to the LP, which is then resolved via the dual simplex method. This new constraint is chosen so as to eliminate x^o from the feasible region without excluding any feasible integral points. Proper construction of these cuts, using data available from the current simplex tableau, ensures that after a finite number of iterations the optimal solution to the augmented LP will be integral, and hence will also solve the ILP. Recent work has been devoted to determining "better" cuts, i.e., additional constraints which will more efficiently lead to the integer solution (see Balas [1974] and Balas and Jeroslow [1975]).

An important characteristic of the cutting plane method is that while dual feasibility is maintained throughout, no feasible integral solution is generated until the final iteration. Since a large number of cuts are often required to reach an integral solution, the inability to terminate the procedure prematurely with a feasible solution could be a serious drawback if computational resources are limited. Indeed, computational experience with cutting plane methods has been disappointing (Trauth and Woolsey [1969]).

The group theoretic approach to integer programming is based on a transformation first proposed by Gomory [1965]. This reformulation begins with the addition of slack variables to the ILP, so it can be written in the following form:

$$\begin{aligned} &\text{maximize} && cx \\ (1)\qquad &\text{subject to} && Ax = b \\ & && x \geq 0 \text{ , integer} \end{aligned}$$

For this discussion, which follows that of Shapiro [1968a], assume that A, b, and c are integral, and that the LP relaxation of (1) has an optimal solution with associated basis B. By partitioning x into basic variables x_B and nonbasic variables x_R, we can transform (1) to

$$\begin{aligned} &\text{maximize} && c_R x_R + c_B x_B \\ (2)\qquad &\text{subject to} && Rx_R + Bx_B = b \\ & && x_R, x_B \geq 0 \text{ , integer} \end{aligned}$$

Solving for x_B and dropping the constant $c_B B^{-1} b$ from the objective function, we obtain

$$\text{maximize} \quad \bar{c} x_R$$

$$\text{(3)} \qquad \text{subject to} \quad x_B = B^{-1} b - B^{-1} R x_R$$

$$x_R,\ x_B \geq 0 \ , \ \text{integer}$$

where $\bar{c} = c_R - c_B B^{-1} R \leq 0$ since B is dual feasible. Now x_B will be integral if and only if $B^{-1} R x_R \equiv B^{-1} b \pmod 1$, or equivalently, if their fractional parts are equal. Assuming for the moment that

$$\text{(4)} \qquad B^{-1}(b - R x_R) \geq 0 \ ,$$

the nonnegativity conditions on x_B can be dropped, and the problem can then be stated solely in terms of x_R. Clearing fractions by multiplying through by $D = |\det B|$ yields the group theoretic formulation (GTP):

$$\text{maximize} \quad \bar{c} x_R$$

$$\text{(GTP)} \qquad \text{subject to} \quad Q x_R \equiv q \pmod D$$

$$x_R \geq 0 \ , \ \text{integer}$$

where $Q = D(B^{-1} R - [B^{-1} R])$, $q = D(B^{-1} b - [B^{-1} b])$, and $[a]$ denotes the integer part of a.

Problem (GTP) can be solved using a shortest route algorithm (see Denardo and Fox [1979]). If the solution x_R is such that (4) holds, we are done. Otherwise an enumerative type of search, involving the solution of (GTP) for several different dual feasible bases B, must be executed (Shapiro [1968b], Gorry and Shapiro [1971]).

The term "implicit enumeration" is descriptive of a wide range of integer programming algorithms, not all of which can be surveyed in the limited space here. Attention will be focused on branch-and-bound methods, since an LP-based branch-and-bound technique was chosen for the implementation of the staircase algorithm proposed in Chapter 2. However there are two approaches that should be mentioned at this point, namely the bound-and-scan algorithm of Hillier [1969] and a new dynamic programming technique developed by Cooper and Cooper [1978].

Hillier's bound-and-scan algorithm requires as input a "good" feasible integer solution x', perhaps generated by an heuristic procedure (e.g., Faaland and Hillier [1979]), and an optimal (non-integer) solution x^o to the LP relaxation of the ILP. By adjoining the constraints $cx \geq cx'$ to the set of binding constraints at x^o, (i.e., those constraints which are satisfied with equality at x^o)*, a simplex is defined which must contain an optimal integer solution. This simplex, or a suitably transformed equivalent (as used in the accelerated bound-and-scan algorithm of Faaland and Hillier [1975]), is efficiently searched by an enumeration routine which, in part, "scans" over the values of a subset of the variables. Tight conditional bounds expedite this search procedure.

*In case of degeneracy, a subset of the binding constraints is used.

Cooper and Cooper have recently demonstrated that dynamic programming methods can be applied to integer programming problems without necessitating huge storage capacities. Their algorithm searches hyperplanes defined by $cx = k$, for k an integer, $k \leq cx^{o}$. The target objective value k is reduced only after the corresponding hyperplane has been shown to contain no feasible integral points. An analytic representation of the optimal solution to a dynamic programming formulation of the problem allows the algorithm to proceed rapidly with a minimal amount of storage space.

Let us turn now to a description of branch-and-bound methods. First proposed by Land and Doig [1960], these algorithms have been modified, improved, and refined for the last twenty years. There are several excellent surveys of this subject, including those of Lawler and Wood [1966], Geoffrion and Marsten [1972], and Geoffrion [1976].

Consider the integer linear programming problem with bounds, given by

$$\begin{aligned} &\text{maximize} && cx \\ (5)\qquad &\text{subject to} && Ax \leq b \\ & && L_j \leq x_j \leq U_j \ , \ x_j \ \text{integer}, \quad j = 1, \ldots, n \end{aligned}$$

This is a more general formulation than that given in Section 1, provided L_j and U_j are not restricted to finite values.

Initially, the branch-and-bound algorithm attempts to solve (5), commonly by applying the simplex method to its LP relaxation. Assuming

that (5) is feasible and an optimal solution is not immediately discovered, the set of feasible solutions to (5) is partitioned by the creation of two new problems, differing from the original only in the bounds on one of the variables, say x_k. The bounds are adjusted to $L_k \leq x_k \leq U_k'$ for one subproblem and $U_k' + 1 \leq x_k \leq U_k$ for the other, where U_k' is an integer such that $L_k \leq U_k' \leq U_k$, $L_k \neq U_k$. Each of these subproblems is placed on a list of nodes, i.e., problems remaining to be solved. Associated with each node i is a bound, objbnd(i), on the optimal objective value attainable for the corresponding problem. The variable x_k is called the branch variable, and the entire process is called a branch.

A skeleton version of a branch-and-bound procedure would invariably contain the following steps:

1. Initialize the node list to include only (5).
2. Remove a problem from the node list; attempt to solve it.
3. If the current problem is "fathomed", discard it and go to step 2. Otherwise continue to step 4.
4. Choose a branch variable x_k, branch, go to step 2.

The procedure halts when it returns to step 2 and finds the node list empty. If the LP relaxation of (5) has a bounded optimum, finite termination can be guaranteed.

Ordinarily, the choice of branch variable x_k is based on penalties which are computed at each iteration. These penalties are a lower bound on the change in the objective value that will result

from branching on x_k. The up (respectively, down) penalty represents the minimal degradation in objective value resulting from increasing (respectively, decreasing) variable j from its current non-integral value x_j to $[x_j] + 1$ (respectively, $[x_j]$).

Fathoming (discontinuance of the search for a solution to the current subproblem i) occurs in three cases:

1. Subproblem i is found to be infeasible.
2. objbnd(i) $\leq$ objective value of the incumbent $\bar{x}$.*
3. x', an optimal solution to subproblem i, is found.

The incumbent $\bar{x}$ is the best integer solution encountered so far. In case 3, if cx' is greater than the incumbent objective value $c\bar{x}$, x' replaces the incumbent. When the algorithm terminates, the incumbent is an optimum for (5) (unless no feasible point was found, in which case (5) is infeasible).

The above outline can give rise to a large assortment of algorithms. Some of the options which have been implemented pertain to orderings for removal of problems from the node list and methods for obtaining objbnd(i). Fathoming techniques, including surrogate constraint methods (Glover [1968], Geoffrion [1969]), can be found in abundance in the literature. In addition, judicious choice of the branching variable can greatly enhance the efficiency of this approach. Hence devices such as pseudocosts have been developed (see Forrest, et al. [1974], Gauthier and Ribiere [1977]).

*Until the first solution is found, the incumbent objective value is taken to be $-\infty$.

Computational experience with branch-and-bound algorithms has been relatively encouraging; they dominate practical usage today. Partially for this reason, this approach was chosen for the implementation of the staircase integer programming technique. As this method is developed in Chapter 3, one specific instance of the general framework described above will be examined in detail.

As reported in Garfinkel and Nemhauser [1973], computational results with the integer programming algorithms characterized in this section have fallen far short of expectations. Unlike linear programming problems, for which a theoretical polynomial time bound exists (Khachian's algorithm, see Gacs and Lovasz [1979]), there is no known method for solving ILPs with solution times that do not grow exponentially with the size of the data. Indeed, it appears rather unlikely at this stage that any such algorithm will ever be found, since the ILP belongs to a class of difficult problems known by the name NP-complete (see Aho, et al. [1974]). Even the average case behavior of these techniques is not of polynomial time complexity, (see e.g., Ibaraki [1977]), as seems to be the case in practice for the simplex method. However, current research in the field of expected behavior of ILP algorithms may lead to the development of solution methods with improved average case behavior (Karp [1977], Lenstra and Rinnooy Kan [1978]).

3. Decomposition Methods

Decomposition methods, in general, attack a problem using a divide-and-conquer strategy. Large problems are broken up in accordance with their special structure into smaller component subproblems which are treated separately. Dependencies between subproblems are usually

handled by a so-called master program which coordinates them and allows them to exchange information. These techniques have been receiving a great deal of attention recently because of the huge size of the models being build to solve current problems, especially in the energy field.

Despite the remarkable improvement in speed of computers over the last two decades, large unstructured integer programming problems remain virtually intractable due to the tendency for solution times to grow exponentially with the number of variables. However, this exponential growth rate implies that the smaller subproblems can usually be repeatedly solved in a small fraction of the time needed for the original undecomposed problem. A similar observation holds for linear programming because the simplex method, in practice, solves a problem in time roughly proportional to the cube of the number of rows. Furthermore, the size of practical linear programming problems can easily exceed available in-core (high speed) memory capacity, thus slowing down execution time dramatically. By solving (several) smaller subproblems, the need to access external storage devices will be reduced, along with computation time.

Motivated by the desire to quickly solve large, structured linear programming problems, Dantzig and Wolfe [1960] developed a specialized algorithm. Their method, known as the Dantzig-Wolfe decomposition principle, was designed to solve a linear programming problem with <u>block-angular</u> structure, i.e., a problem for which one (or several) rows link otherwise unrelated subproblems. In the formulation below, submatrices A_i, $i = 1, \ldots, k$, form the linking rows of a block-angular constraint matrix with k subproblems.

BLOCK-ANGULAR PROBLEM

$$
\begin{aligned}
\text{maximize} \quad & c_1x_1 + c_2x_2 + \quad \cdots \quad + c_kx_k \\
\text{subject to} \quad & A_1x_1 + A_2x_2 + \quad \cdots \quad + A_kx_k \leq b_0 \\
& B_1x_1 \qquad\qquad\qquad\qquad \leq b_1 \\
& \qquad B_2x_2 \qquad\qquad\quad \leq b_2 \\
& \qquad\qquad \cdot \qquad\qquad\quad \cdot \\
& \qquad\qquad\quad \cdot \qquad\quad \cdot \\
& \qquad\qquad\qquad \cdot \qquad\quad \cdot \\
& \qquad\qquad\qquad\qquad B_kx_k \leq b_k
\end{aligned} \tag{6}
$$

$$x_i \geq 0, \quad i = 1, \ldots, k$$

Consider first the constraints of the j^{th} subproblem:

$$B_jx_j \leq b_j, \qquad x_j \geq 0 \tag{7}$$

Assuming that the feasible region defined by these constraints is bounded,* any feasible x_j can be written as a convex combination of the extreme points $w_j(1), w_j(2), \ldots, w_j(E_j)$ of (7). Problem (6) can then be reformulated in terms of these extreme points, resulting in the full master problem (M):

*For a discussion of the modifications necessary for the more general case, see Dantzig and Wolfe [1960].

MASTER PROBLEM (M)

$$\text{maximize} \quad \sum_{i=1}^{E_1} \rho_1(i)w_1(i) + \sum_{i=1}^{E_2} \rho_2(i)w_2(i) + \cdots + \sum_{i=1}^{E_k} \rho_k(i)w_k(i)$$

$$\text{(8)} \quad \text{subject to} \quad \sum_{i=1}^{E_1} \lambda_1(i)\alpha_1(i) + \sum_{i=1}^{E_2} \lambda_2(i)\alpha_2(i) + \cdots + \sum_{i=1}^{E_k} \lambda_k(i)\alpha_k(i) \leq b_0$$

$$\sum_{i=1}^{E_1} \lambda_1(i) = 1$$

$$\sum_{i=1}^{E_2} \lambda_2(i) = 1$$

$$\text{(9)} \qquad \vdots$$

$$\sum_{i=1}^{E_k} \lambda_k(i) = 1$$

$$\lambda_j(i) \geq 0 \quad \text{for} \quad j = 1, \ldots, k \quad \text{and} \quad i = 1, \ldots, E_j$$

where

$$\rho_j(i) = c_j(i)\lambda_j(i)$$

and

$$\alpha_j(i) = A_j w_j(i) \quad \text{for all} \quad i,j.$$

Although this transformed problem has fewer rows than (6), in general it contains many more columns. If it were necessary to generate all of these columns, this method would not be very efficient. Fortunately

the revised simplex method can be used, so that at each iteration only the column of (M) which is about to enter the basis has to be computed. That particular column will be the one with lowest reduced cost, provided this cost is negative.* Letting π be the current simplex multipliers (dual variables) associated with rows (8) of (M), we can find the appropriate column by solving the subproblems S_j, for $j = 1, \ldots, k$, where S_j is defined by:

$$\begin{aligned} & \text{minimize} && \pi A_j x_j \\ (S_j) \qquad & \text{subject to} && B_j x_j \leq b_j \\ & && x_j \geq 0 \end{aligned}$$

The solution of each S_j generates an extreme point $w_j(i)$ of (7) with corresponding reduced cost $\pi A_j w_j(i)$; by minimizing this over all j, the entering column is produced. Reduced to basic and entering columns only, problem (M) is called the restricted master problem.

This decomposition approach bears a strong resemblance to decentralized planning techniques within a multi-divisional corporation (Dantzig [1963]). Regarding the subproblems S_j as intra-divisional optimization problems for generating production proposals, the master problem (M) can be thought of as a managerial problem of coordinating the various divisions while keeping within the company resources defined by constraints (8). Proposals are passed from each division to the

*If all reduced costs are nonnegative, the current solution is optimal.

manager. After evaluating these recommendations, he sends back new prices (π) on goods, labor and materials. Thus the divisions are motivated to submit additional proposals which can be mixed with their previous ones to form a company-wide feasible, optimal strategy.

Application of these methods to LPs with a staircase constraint matrix (as in Figure 1) was first proposed by Dantzig and Wolfe in the aforementioned paper, and later by Dantzig [1963]. Since then, nested decomposition methods have been examined by several researchers, including Glassey [1973] and Ho and Manne [1974]. Briefly, this approach begins by decomposing the staircase problem into a period 1 "subproblem", and a "master" problem consisting of constraints from periods 2, 3, ..., T. Recursive decomposition of the master problem yields T problems SP_j, $j = 1, \ldots, T$, with the property that SP_j behaves as a subproblem with respect to SP_k for $k > j$ and as a restricted master problem for SP_i for $i < j$.

In most versions of nested decomposition, the algorithm proceeds by solving the subproblems backward in time, from period T to period 1. During each such cycle, prices are calculated to pass one period backward in time and proposals are generated to be passed one period forward on the next cycle. Although preliminary computational experience has been somewhat disappointing, Ho [1977] has recently obtained good results using multi-proposal generation (see Ho [1974]) on a particular class of moderate-sized staircase test problems. Furthermore, promising research is currently being carried out on a variation of the nested decomposition algorithm that moves forward and backward in time, working from a dual standpoint (Abrahamson [1980]).

Many other techniques for solving staircase LPs have been developed; a fairly complete list is given by Madsen [1977]. The majority of these methods try to take advantage of the staircase structure inherited by the basis matrix in order to simplify calculation and updating of the basis inverse (see Heesterman and Sandee [1965], Saigal [1966], Dantzig [1973], and Wolmer [1979]). Although computational experience with these algorithms is severely limited, Fourer [1980] has thoroughly tested, with great success, an adaptation of the simplex method for solving staircase LPs using sparse matrix and partial pricing techniques.

The first application of decomposition principles to integer programming was Benders Decomposition, a technique designed to solve mixed variable programming problems, especially mixed integer programs (see Benders [1962] and Geoffrion and Graves [1974]). The basic idea of this approach is to convert the problem into an (almost) pure integer master program, usually with many more rows. These constraints are generated, as needed, by the solution of an appropriate dual LP. Thus the concept of a restricted master problem is carried over from Dantzig-Wolfe decomposition.

Exploitation of special structure in an ILP began with the work of Schrage [1973] on solving pure-integer problems with either staircase or dual angular structure. A problem is said to have the latter structure if the constraint matrix has a relatively small set of columns linking otherwise independent subproblems, as in Figure 3. Note that the dual LP has a block-angular structure.

In order to solve this problem, Schrage's strategy is to execute a branch-and-bound search, always branching first on linking variables (those variables corresponding to the linking columns formed by

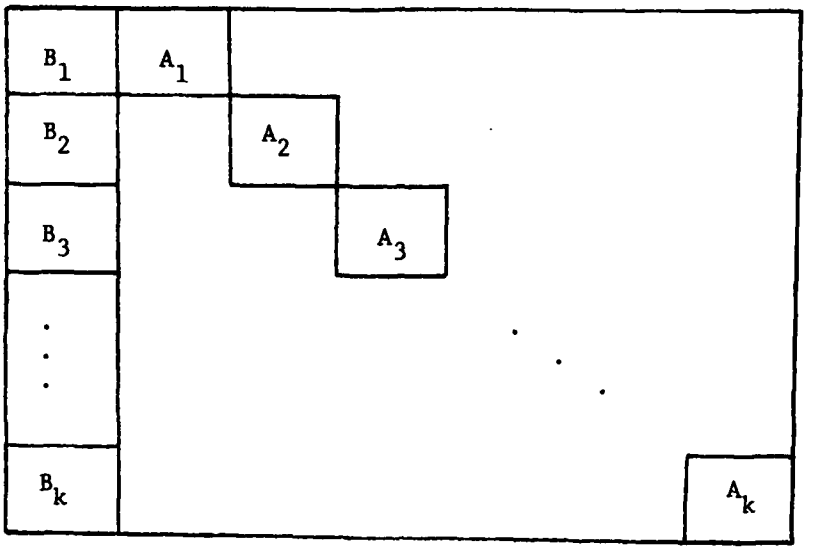


Figure 3: Dual Angular Matrix

submatrices B_1, B_2, ..., B_k, in Figure 3). After all such variables are fixed at integral values, the problem decomposes naturally into its component subproblems, which may then be solved independently.

For a staircase matrix with k diagonal blocks, the crucial observation is that once the variables from period $[k/2] + 1$ have been fixed at an integral value, the matrix decomposes into two independent staircase submatrices with $[k/2]$ and $[(k - 1)/2]$ blocks. Applying this idea recursively to the component staircase problems produces a branching order that ultimately leads to maximal decomposition of the problem.

Limited computational experience with this staircase algorithm has demonstrated that it is extremely sensitive to the number of linking columns. Moreover it has yet to be tested on problems of even moderate size.

As a modification to Schrage's approach to dual angular problems, Reardon [1974] suggested decomposition into independent subproblems from the start of the search, that is, before any of the linking variables are fixed. This decomposition is achieved by temporarily relaxing the implicit constraint that each linking variable must attain the same value in the solution to each subproblem. As in Schrage's method, the linking variables are considered first. As soon as a branch is taken on a linking variable in one subproblem, that same branch is repeated in every subproblem. Thus the implicit constraint is reimposed whenever a branch is taken. True decomposition occurs when all the linking variables have been set to an integral value. Because the problem is treated as if decomposed from the start, however, the computational expense of pivoting in the large tableau associated with the original problem is avoided. In addition to Reardon's favorable computational results, application of this method to a specialized integer linear programming problem has proven successful (Kochman and Pollenz [1978]).

Block-angular ILPs have also been solved using decomposition methods. Treating the right-hand side b_0 of the linking rows of (6) as a set of precious resources to be shared among the subproblems, Kochman [1976a] executes a branch-and-bound search of the feasible distributions of b_0. This technique, called resource decomposition, recognizes an allocation $(\beta_1, \beta_2, \ldots, \beta_k)$ as feasible if

$$\sum_{i=1}^{k} \beta_i = b_0 \tag{10}$$

and if for $i = 1, \ldots, k$, there exists an integral x_i such that (11) holds.

$$A_i x_i \leq \beta_i$$

(11) $$B_i x_i \leq b_i$$

$$x_i \geq 0$$

Notice that this transformation of the model has a dual angular structure, with the allocation variables β_i as the linking variables. Using an adaptive version of Reardon's method, Kochman reported excellent computational results for several test problems.

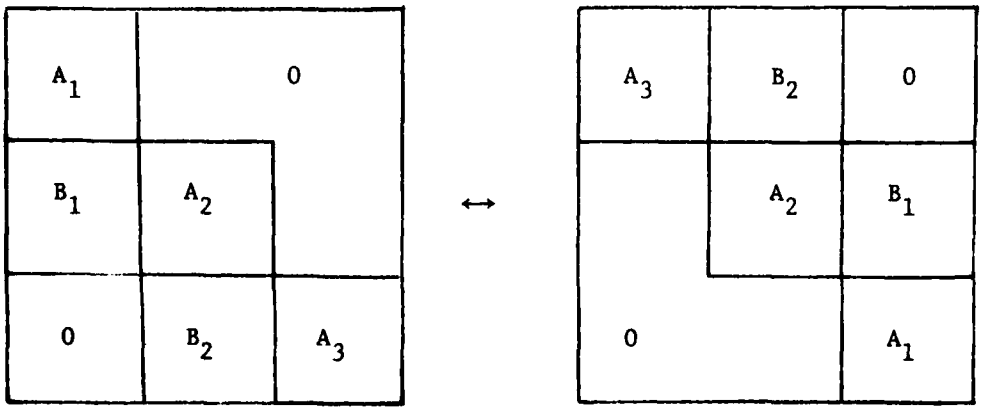
4. Overview

To supplement the background information provided in the preceding sections, several important definitions are presented in the first section of Chapter 2. These concepts are necessary for the development of the staircase decomposition algorithm outlined in the latter part of that chapter. The third chapter contains a detailed description of the procedure, including modifications for some special cases. Objective bounds for fathoming are developed in Chapter 4, along with computational experience on the effectiveness of such bounds. In Chapter 5, the staircase decomposition method is extended to higher order staircase matrices, mixed integer linear programs, and certain nonlinear integer programs. Computational results, including comparison with a branch-and-bound algorithm applied to the undecomposed problem, are presented in Chapter 6. Conclusions and directions for future research are given in the seventh and final chapter.

CHAPTER 2: A STAIRCASE DECOMPOSITION ALGORITHM

1. The Staircase Structured Problem

A staircase matrix A (see Figure 1) is formally defined by a partition of the m rows into T disjoint subsets and the n columns into T (different) disjoint subsets. Rows (columns) which belong to the k^{th} subset of this partition are said to be constraints (activities) of the k^{th} period. Furthermore, the matrix is defined to be a lower staircase matrix if these periods are ordered so that constraints of period k have nonzero coefficients only in columns of periods k and k - 1. Alternatively, if the period k constraints involve only columns from periods k and k + 1, the matrix is called upper staircase. Since any upper staircase matrix can be permuted to lower staircase form by simply reversing the order of the partition (see Figure 4), these two characterizations are equivalent. Therefore, without loss of generality, we will henceforth assume a staircase matrix to be in lower staircase form.



Lower Staircase Form Upper Staircase Form

Figure 4: Equivalence of Staircase Forms

The j^{th} variable of period t will be denoted by $x_t(j)$. As defined in Chapter 1, $x_t(j)$ is a non-linking or local variable if it is associated with a column of A which has nonzero elements only in the rows of period t. Linking variables appear with nonzero coefficients in constraints of two adjacent periods. Linking and non-linking constraints can be defined in a similar manner.

The version of the staircase integer programming problem that will be solved by the decomposition method to be developed here can be formulated as follows:

$$
\begin{aligned}
&\text{maximize} && c_1x_1 + c_2x_2 + \cdots + c_{T-1}x_{T-1} + c_Tx_T \\
&\text{subject to} && A_1x_1 && \leq b_1 \\
&&& B_1x_1 + A_2x_2 && \leq b_2 \\
&\text{(P)} && \qquad\qquad \ddots && \vdots \\
&&& \qquad\qquad B_{T-1}x_{T-1} + A_Tx_T && \leq b_T
\end{aligned}
$$

$$L_t(j) \leq x_t(j) \leq U_t(j), \text{ for } j = 1, \ldots, n_t, \text{ and } t = 1, \ldots, T$$

$$x_t \text{ integer}, \quad t = 1, \ldots, T$$

where the lower and upper bounds $L_t(j)$, $U_t(j)$ are finite. In Section 3 we will explore the nature of the bounding restriction and consider methods for generating such bounds when none are given. Note however, that no restrictions are placed on the coefficients of the submatrices A_t, B_t, or the vectors b and c except that they be rational.

The decomposition algorithm will operate by solving the subproblems S_t, $t = 1, \ldots, T$, given by:

$$\text{maximize} \quad c_t x_t$$

$$(S_t) \qquad \text{subject to} \quad A_t x_t \leq b_t - B_{t-1} x_{t-1}$$

$$L_t \leq x_t \leq U_t$$

$$x_t \quad \text{integer}$$

where $B_o = 0$.

The term $B_{t-1}x_{t-1}$ has been moved to the right-hand side of the constraints because it is assumed that by the time the period t subproblem is solved, the values of the period $t - 1$ variables have already been set. Thus both the optimal solution and the set of feasible solutions to subproblem S_t is dependent upon the solution to subproblem S_{t-1}. In fact, a more accurate notation would reflect this conditional nature of subproblem t, as in $S_t(x_{t-1})$. Nevertheless, for the sake of brevity the notational dependence on x_{t-1} will be dropped unless special emphasis of this dependence is deemed appropriate.

A vector $x = (x_1, x_2, \ldots, x_k)$ such that x_t is feasible for S_t, for $1 \leq t \leq k$, will be called a <u>partial solution</u> for (P). If there exists a vector $x' = (x_1, \ldots, x_k, x_{k+1}, \ldots, x_T)$ such that x_t is feasible for S_t, for $t > k$, then x' is both a <u>completion</u> of x and a <u>complete solution</u>. This terminology differs from conventional definitions in that a completion is required to be feasible.

2. Outline of the Decomposition Method

Underlying the decomposition algorithm is a basic search procedure for enumerating all the feasible solutions to each subproblem S_t for a fixed right-hand side. This search procedure can utilize any of a number of integer programming solution techniques; the advantages and disadvantages of some of these methods will be discussed in the next chapter. All that is required of the search procedure is that it generate all feasible solutions to S_t in some systematic way (without repetition), so that the search can be suspended at some point and resumed without difficulty later.

The overall structure of the algorithm resembles that of an enumerative search over the set F, which is defined by:

$$F = \{(x_1, x_2, \ldots, x_T) \mid x_t \text{ is feasible for } S_t(x_{t-1}),\ t = 1, \ldots, T\}.$$

Clearly, the set of feasible solutions to (P) is a subset of F. If we consider the subproblems S_t in their proper time sequence, (i.e., from 1 to T), it becomes apparent that an element $(x_1, x_2, \ldots, x_T)$ of F also satisfies the constraints of (P). Thus the two sets are equivalent, and problem (P) can be solved by finding that element of F which maximizes the objective value of (P).

The algorithm begins by searching for a feasible solution x_1 to subproblem S_1. If S_1 is infeasible, then so is (P), since the constraints of S_1 are identical to the period 1 constraints of (P). Suppose then that S_1 is feasible. Once the search routine finds a feasible point x_1, the search is temporarily halted, and a new search

is begun for solutions to subproblem S_2 given x_1. This process of solving subproblems and moving forward to the next time period continues until the current solution path is fathomed. As in a standard branch-and-bound search (see Section 2 of Chapter 1), fathoming while solving S_t occurs in these three cases:

1. All feasible solutions to S_t (if any) have been explored for possible completions of the current partial solution $(x_1, x_2, \ldots, x_{t-1})$.*
2. An objective value bound proves that no unexplored completion of $(x_1, x_2, \ldots, x_{t-1})$ can result in a better solution than the incumbent.
3. $t = T$, and an optimal solution to S_T has been encountered.

Upon fathoming in cases 1 and 2, the algorithm backtracks to the previous subproblem S_{t-1}, and continues the search procedure from the point at which it left off. In case 3, the objective value of the newly generated complete solution x must be greater than the incumbent objective value (otherwise we would have fathomed due to criterion 2). After replacing the incumbent with x, the algorithm backtracks to the last time period $T - 1$ to seek a better completion of $(x_1, x_2, \ldots, x_{T-2})$.

*Each time a forward step to period t is taken, a new search for feasible solutions to S_t is begun. Thus a subproblem solution may be examined several times along different solution paths.

This decomposition algorithm, in its most general form, can be summarized in 6 steps:

Step 1. Initialization: Input all data. Set $k = 1$.

Step 2. Search Procedure: Find an unexplored, "promising" feasible solution to S_k. If no feasible solution x_k that can yield an improved complete solution exists, go to Step 5. Otherwise go to Step 3.

Step 3. Forward Step: If $k = T$ go to Step 4. Otherwise, suspend and store the current search procedure position for S_k, alter the right-hand side of S_{k+1} according to the newly set value of x_k, increase k by 1, and return to Step 2.

Step 4. Improved Complete Solution: Record the solution $(x_1, x_2, \ldots, x_T)$ as the new incumbent. Go to Step 2.

Step 5. Backtrack Step: If $k = 1$, go to Step 6. Otherwise decrement k by 1, restore the search procedure of S_k and go to Step 2.

Step 6. Termination: Halt. The incumbent is an optimal solution, unless no complete solution has been found. In that case, no feasible solution to (P) exists.

A feasible solution is considered "promising" in Step 2 if the objective value bound on the associated partial solution $(x_1, \ldots, x_k)$ is greater than the incumbent objective value $\bar{z} = c\bar{x}$. A very crude objective bound is given by (12).

$$(12)\quad z \leq \sum_{t=1}^{k} c_t x_t + \sum_{t=k+1}^{T} \sum_{j=1}^{n_t} c_t(j)(\delta_{t,j} L_t(j) + (1 - \delta_{t,j}) U_t(j))$$

where

$$\delta_{t,j} = \begin{cases} 0 & \text{if } c_t(j) \geq 0 \\ 1 & \text{if } c_t(j) < 0 \end{cases}$$

Although this bound is valid, and is perhaps useful for exposition purposes, it is not very effective in practice. Improved objective bounds, which greatly speed the fathoming procedure, are developed in Chapter 4.

Termination, Step 6, occurs when all feasible solutions to S_1 have been examined for completions. Finite termination is highly desirable if we actually intend to implement this algorithm. To show this property, let M be the number of complete solutions examined by the procedure described above. Define N_t by:

$$(13)\qquad N_t = \prod_{j=1}^{n_t} (U_t(j) - L_t(j) + 1)$$

Then we have the following

<u>Theorem</u>: $M \leq \prod_{t=1}^{T} N_t$.

Proof:

At most, each subproblem S_t has N_t feasible solutions. Now, if the number of periods, T, is 1, then M is trivially bounded by N_1. Inductively assume that if $T \leq p$, then the theorem is true.

Next suppose $T = p + 1$. For each feasible solution x_1 to S_1 the algorithm will consider all feasible solutions $(x_2, x_3, \ldots, x_T)$ to the remaining subproblems (where S_2 is understood to have right-hand side $b_2 - B_1x_1$). Since the remaining subproblems form a staircase problem with p periods, the induction hypothesis applies, and we obtain

$$M \leq N_1 \prod_{t=2}^{p+1} N_t = \prod_{t=1}^{T} N_t . \tag{14}$$

Thus, by induction, the theorem is true for all T. ⊠

Since we will choose to implement this algorithm with some integer programming search technique that finds all the feasible solutions to a subproblem (without repetition) in a finite amount of time, finite termination of the algorithm is guaranteed.

3. Generating Initial Bounds

The formulation of problem (P) given in Section 1 includes finite upper and lower bounds on the variables. In most applications, lower bounds (usually all zero) are given, but upper bounds are often omitted. Sometimes maximum values are clear from the interpretations or physical meanings of the variables. Whether or not upper bounds

are given or obvious, the staircase decomposition algorithm relies on their existence. Without them, finite termination is not guaranteed.

In this section, we will consider the problem of finding upper bounds for the general integer programming problem (ILP) given below, where the elements of the $m \times n$ matrix A are all rational numbers.

$$
\begin{array}{lll}
 & \text{maximize} & cx \\
\text{(ILP)} & \text{subject to} & Ax \leq b \\
 & & x \geq 0, \quad \text{integer}
\end{array}
$$

Let (LP) denote the LP relaxation of (ILP). The following result (due to Gomory) demonstrates that boundedness of (ILP) is directly related to that of (LP).

Lemma: Suppose (ILP) is feasible. Then the objective value of (ILP) is unbounded if and only if (LP) has unbounded objective value.

Proof:

==>: If (ILP) has unbounded objective value, then clearly any relaxation of (ILP) cannot have a finite maximum. Thus, the objective value of (LP) would also be unbounded.

<==: Let $\hat{x}$ be feasible for (ILP). Suppose that (LP) is dual infeasible, although primal feasible. Applying the simplex method results in a basic feasible solution x, basis B, and a ray r of the form:

$$r_j = \begin{cases} 1, & j = k \\ -\bar{a}_{ik}, & x_j \text{ is basic in row } i \\ 0, & \text{otherwise} \end{cases}$$

Here x_k is some nonbasic variable and $\bar{a}_{ik} = (B^{-1}A)_{ik} \leq 0$.

The ray r has rational elements because A is assumed to be a rational matrix. Therefore, for some constant $\alpha > 0$ sufficiently large, all components of αr will be integral. Since r is a ray, each element of $Ar \leq 0$ and $cr > 0$. Thus $\hat{x} + i\alpha r$, for $i = 1, 2, \ldots$, is a sequence of points feasible for (ILP), with $\lim_{i \to \infty} c(\hat{x} + i\alpha r) = \infty$.

⊠

This lemma does not necessarily hold for the case of irrational data (see Kaneko [1974]).

According to the lemma, if (LP) is dual infeasible, (ILP) is either infeasible or has unbounded objective value, and hence is of little further interest. Assume then that (LP) has optimal objective value $z_o < \infty$. To partly answer the question of when upper bounds on the variables do exist, we have the following three theorems.

Theorem 1: If $c_j \geq 0$ for all $j = 1, \ldots, n$, then for all k such that $c_k > 0$, $x_k \leq [z_o/c_k]$.

Proof:

Suppose $y_k > (z_o/c_k)$. Then any solution $x \geq 0$ such that $x_k = y_k$ would have objective value greater than z_o. Thus x could not be feasible for (LP), and therefore it is also infeasible for (ILP).

With the addition of the integrality restriction on x_k, the desired bound is obtained. ⊠

Theorem 2: If $c_j \leq 0$ for all $j = 1, \ldots, n$, and $\hat{x}$ is a feasible solution to (ILP), then for all k such that $c_k < 0$, $x_k \leq [c\hat{x}/c_k]$.

Proof:

Suppose $y_k > c\hat{x}/c_k$, so that $c_k y_k < c\hat{x}$. Then any solution $x \geq 0$ such that $x_k = y_k$ would have objective value $cx \leq c_k x_k < c\hat{x}$. Thus x could not be an optimal solution to (ILP).

Adding the integrality restriction on x_k results in the upper bound given above. ⊠

Theorem 3: If for some row i, $A(i,j) \geq 0$ for all $j = 1, \ldots, n$, then for all k such that $A(i,k) > 0$, $x_k \leq [b_i/A(i,k)]$.

Proof:

If $x_k > b_i/A(i,k)$, then the constraints $\sum_{j=1}^{n} A(i,j)x_j \leq b_i$ and $x \geq 0$ cannot be satisfied simultaneously. After taking into consideration the integrality restriction on x_k, the result is proven. ⊠

Thus upper bounds are guaranteed to exist if any of the following conditions hold:

(1) all costs are strictly positive;

(2) all costs are strictly negative and a feasible solution is known;

(3) at least one row of A has all coefficients strictly positive.

Upper bounds on some of the variables may still be available from Theorems 1, 2 and 3, even if none of the three conditions above holds. In fact, after upper bounds U_j, for $j \in U$, have been discovered for a subset U of the variables, the hypotheses of the three theorems can be slightly weakened. The following corollary to Theorem 3 is illustrative of the kind of result easily obtained from any of the three theorems.

Corollary: If for some row i, $A(i,j) \geq 0$ for all $j \notin U$, then for all $k \notin U$ such that $A(i,k) > 0$, then

$$x_k \leq (b_i - \sum_{j \in U} \gamma_j A(i,j) U_j)/A(i,k) \tag{15}$$

where

$$\gamma_j = \begin{cases} 0, & A(i,j) \geq 0 \\ 1, & A(i,j) < 0 \ . \end{cases}$$

For the general case in which no special conditions are placed on the cost coefficients or the constraint matrix, no guarantees can be given that upper bounds will exist on all variables. If no such maxima are given or readily discernable, various procedures can be executed in an attempt to generate the required bounds.

For example, each variable x_k can be maximized subject to the constraints of (LP) and the known upper bounds. If this linear program has a finite optimum, its objective value will be an upper bound on x_k. Unfortunately, there is no guarantee that such an upper bound exists.

The heuristic procedure of repeatedly trying to find bounds by applying the corollaries of Theorems 1, 2 and 3 and solving the linear program mentioned above may fail to find maximum values for all the variables. Even in this case, the staircase decomposition algorithm (in a slightly modified form) is likely to terminate in a finite amount of time, particularly when the problem is feasible with a bounded optimum. However, finite termination is not guaranteed, and another solution method might be appropriate under these circumstances.

CHAPTER 3: A SPECIFIC IMPLEMENTATION OF THE ALGORITHM

1. Subproblem Solution Methods

Finite termination, while necessary, is not sufficient to ensure algorithmic efficiency. If all the numerous solutions to (P) had to be examined explicitly by the algorithm described in the preceding chapter, a lot of computer time would be wasted. This is avoided by generating good objective function bounds for use in fathoming (see Chapter 4). Another important factor in computational speed is how quickly the search procedure can find all the interesting feasible solutions to each subproblem. Also, the order in which these feasible points are discovered can have a substantial impact on the total number of partial and complete solutions explored.

Several different integer programming techniques could be used for the search routine. For example, the dynamic programming hyperplane search algorithm of Cooper and Cooper (see Chapter 1) could easily be modified to generate all feasible solutions to a subproblem. Due to the nature of the hyperplanes used in this method, these solutions would be generated in order of decreasing objective value. This "greedy" approach, while most efficient in a local sense, would probably prove to be far from a globally optimal search strategy for the majority of staircase problems. A good search strategy must take into consideration the effect of the subproblem solution on subsequent (future) subproblems; it must in some sense "look ahead".

Another possible search procedure would make use of cutting plane methods. Although cutting planes are used to generate only an optimal integral solution to the ILP, the method can be extended to the case in question.

Proposition: A variant of the cutting plane method can be used to find all feasible solutions to a bounded ILP.

Proof:

For convenience, assume A, b are integral.* Also, transform the problem, if necessary, so that all lower bounds are 0.

Apply the cutting plane method to the ILP to obtain an optimal integral solution y. Since y is an extreme point solution to the last augmented LP, there is associated with y a basis B, and a set of nonbasic variables N. Add the following constraint to the augmented LP:

$$\sum_{j \in N} x_j \geq 1 \tag{16}$$

Clearly, y violates this constraint. Furthermore, consider any feasible integral point x which also violates this constraint. Since $x \geq 0$ and x is integral, we must have $x_j = 0$ for all $j \in N$. However, the unique solution to the constraints with the remaining (independent) set of variables is y. Thus the hyperplane (16) eliminates y without excluding any other feasible integral points. ☒

Note that the cutting plane approach described above also finds solutions in order of decreasing objective value. As mentioned before, this is not particularly desirable. However it is possible that the search order could be perturbed in a useful way, by adjusting the cost coefficients to reflect the impact of each variable on the optimal solution of the next subproblem.

*If A, b are rational, the right-hand side of (16) must be adjusted to a sufficiently small positive ε.

A far more serious drawback is the large amount of setup time necessary to find the very first feasible subproblem solution, even after the problem has been solved previously with a different right-hand side. For a typical problem, each subproblem (after the first) will be solved many times for many different values of the right-hand side. Thus a method which has a short initial setup time, or which can be modified to reduce subsequent setup times after the initial subproblem solution, is highly desirable. Group theoretic methods and the accelerated bound-and-scan algorithm both seem disadvantageous because they do not have either of these properties.

After solving an ILP with a cutting plane algorithm, parametric changes in the right-hand side can be handled by appropriate modifications in the right-hand sides of the Gomory cuts (see Holm and Klein [1978]). Dual feasibility is always maintained by this procedure, but additional cuts may be necessary to reestablish primal feasibility. Limited computational experience suggests that this parametric version of the cutting plane method is more efficient than solving the problem from scratch. However, it does require additional storage space and computation of a significant number of extra vector products.

Reoptimization after a change in the right-hand side is relatively easy for the simplex method. In fact, dual simplex reoptimization after a branch is standard procedure for many LP-based branch-and-bound codes. In addition, setup time is relatively brief and the branch-and-bound search order is very flexible. Prices, penalties, and/or pseudocosts can be employed to modify the search order to reflect the structure of subsequent subproblems. Finally, computational experience with this

solution technique has been relatively encouraging; most of the integer programming codes commercially available employ some version of this method. In light of the above, an LP-based branch-and-bound routine was selected for the implementation of this decomposition algorithm.

2. The Algorithm in Detail

It is now possible to elaborate on the description of the staircase decomposition algorithm (SDA) given in Chapter 2. Before proceeding with this characterization, a few definitions are needed. First, let maxc(k) be an upper bound on the objective value of the following problem:

$$\text{maximize} \quad \sum_{t=k+1}^{T} c_t x_t$$

$$\text{subject to} \quad (x_1, x_2, \ldots, x_T) \text{ feasible for (P)}$$

One valid, although inefficient, bound is given by (17), where $\delta_{t,j}$ is as defined in (12).

$$\text{(17)} \quad \text{maxc}(k) = \sum_{t=k+1}^{T} \sum_{j=1}^{n_t} c_t(j)(\delta_{t,j} L_t(j) + (1 - \delta_{t,j}) U(j))$$

Next, let

cumobj = $\sum_{t=1}^{k-1} c_t x_t$ = cumulative objective value for $S_1, S_2, \ldots, S_{k-1}$, along the current solution path;

incval = objective value of the incumbent $\bar{x}$;

$z_k(i)$ = optimal objective value of the LP relaxation of node i (descended from subproblem k);

$GP_k(i)$ = maximum Gomory penalty (see Tomlin [1971]) associated with node i of S_k;

$\hat{b}_k$ = $b_k - B_{k-1}x_{k-1}$ = updated right-hand side of S_k.

With the exceptions noted below in Step 2, the following is a detailed characterization of the SDA. (See Figure 5 for a flow diagram of this method).

Staircase Decomposition Algorithm (SDA)

Step 1. Initialization: Input data. Set $k = 1$, $\hat{b}_1 = b_1$, cumobj = 0. Also let incval = $-\infty$ or some known lower bound on the objective value of (P). Go to Step 2.

Step 2. Bound Generation: Compute maxc(t), for $t = 1, 2, \ldots, T - 1$. (See Chapter 4 for a precise definition of these objective value bounds and a description of the procedure for determining them.) At the same time, calculate some prices $P_t(j)$, $j = 1, \ldots, n_t$, (see Section 3) for use in determining branch strategy. Go to Step 3.

Step 3. Initial Subproblem Solution: Solve the LP relaxation of S_k with right-hand side $\hat{b}_k$. Save the optimum basis inverse for later use in Step 5. Go to Step 4.

Step 4. Branch-and-Bound Search: Start (or continue) an LP-based branch-and-bound search for all promising, feasible solutions to $S_k(x_{k-1})$. This search is guided by penalty calculations,

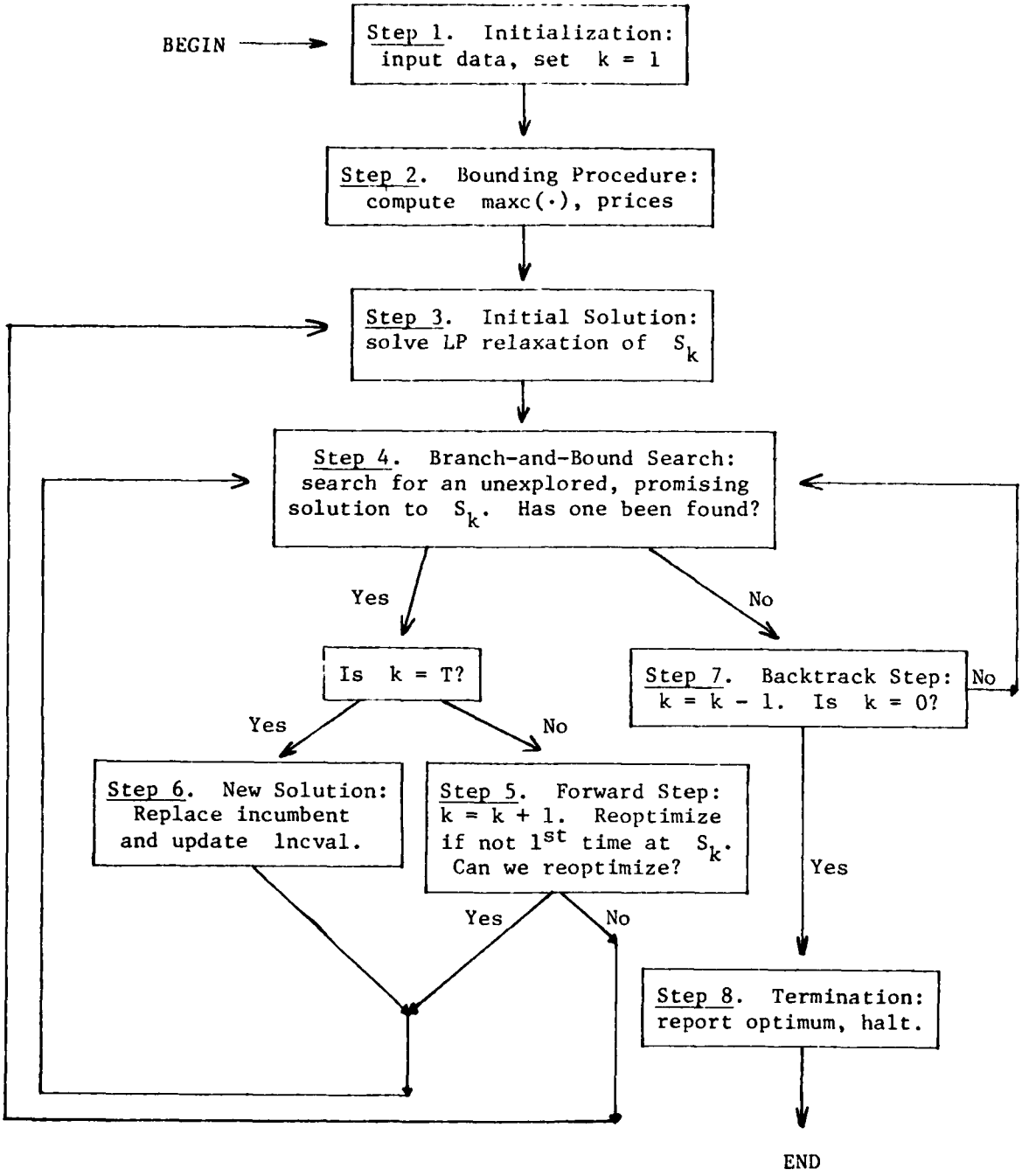


Figure 5: Flow Chart of the Algorithm

although pseudocosts could be substituted. Continue the search until one of the following fathoming conditions holds at the current node i:

(a) the LP relaxation of node i is infeasible.

(b) $\text{cumobj} + \text{FLOOR}(z_k(i) + GP_k(i)) + \text{maxc}(k) \leq \text{incval}$.*

(c) an integral solution x_k has been encountered.

Upon fathoming in cases (a) or (b), backtrack within this subproblem search. If the search then terminates (because all nodes have been explored), jump to Step 7. Assuming (a) and (b) are not satisfied at node i and (c) holds, mark node i as fathomed. Then if $k = T$, (c) implies that $\text{cumobj} + z_T(i) > \text{incval}$. Therefore a new complete solution with improved objective value has been discovered. In this case, go to Step 6. For $k < T$, go to Step 5.

Step 5. Forward Step: Suspend and store the branch-and-bound search of S_k. Set $\text{cumobj} = \text{cumobj} + c_k x_k$ and $\hat{b}_{k+1} = b_{k+1} - B_k x_k$. Set $k = k + 1$. If this is the first time we have tried to solve S_k, go to Step 2. Otherwise, restore the basis inverse saved in Step 3 and reoptimize via the dual simplex method. Go to Step 4.

Step 6. Improved Complete Solution: Replace the incumbent $\bar{x}$ with $(x_1, x_2, \ldots, x_T)$. Set $\text{incval} = \sum_{t=1}^{T} c_t x_t$. Return to Step 4.

*FLOOR(a) = greatest integer not greater than a. This rounding off assumes integral costs.

Step 7. Backtrack Step: Set $k = k - 1$. If $k = 0$, go to Step 8. Otherwise reset cumobj = $\sum_{t=1}^{k-1} c_t x_t$. Restore the search of $S_k(x_{k-1})$ to its most recent state and go to Step 4.

Step 8. Termination: If incval = $-\infty$, report the infeasibility of problem (P). Otherwise report the optimal solution $\bar{x}$ and its objective value incval. Halt.

3. Forward Pricing Methods

While executing the subproblem branch-and-bound search the choice of branch variable and direction (up to the next higher or down to the next lower integral value) should be based at least partly on the coefficients of the entire problem, not just one subproblem. In order to accomplish this, we need some measure of the global effect of changing the value of each variable.

By the time subproblem S_k is solved, the variables of period t, for $t = 1, 2, \ldots, k - 1$, have already been fixed at integral values. Thus, a change in $x_k(j)$ can only affect the optimal objective value (and feasibility) of the present and future, that is, of subproblems S_k, S_{k+1}, ..., S_T. Since the algorithm is currently working on S_k, it has available sufficient information (from the simplex tableau) to compute Tomlin's improved penalties (see Tomlin [1971]). These penalties are used to gauge the first order effects of a change in $x_k(j)$ on the present period k. The effect on subsequent periods is more problematical because the problem is decomposed; the SDA is not allowed to examine more than one period at a time once the iterative portion (starting with Step 3) has begun.

To circumvent this restriction, consider the problem (R_k), which is the LP relaxation of the aggregation of the remaining unsolved subproblems $S_{k+1}, \ldots, S_T$.

$$(R_k) \qquad \text{maximize} \quad \sum_{t=k+1}^{T} c_t x_t$$

$$(18) \qquad \text{subject to} \quad B_{t-1}x_{t-1} + A_t x_t \leq b_t, \quad t = k+1, \ldots, T$$

$$L_t(j) \leq x_t(j) \leq U_t(j), \quad j = 1, \ldots, n_t, \quad t = k, \ldots, T.$$

Suppose we temporarily adjoin to (R_k) the constraints $x_k = L_k$ and solve using the simplex method. Since the majority of practical problems will have $L_t = 0$ and $b_t \geq 0$ for all t, it is plausible to assume that this problem is feasible. Under this assumption, an optimal solution $(x_k^o, x_{k+1}^o, \ldots, x_T^o)$ exists. Furthermore, this vector represents the best that can be achieved in periods $k+1, \ldots, T$, given that each $x_k(j)$ is fixed at its lower bound.

For the augmented problem, let π be the optimal simplex multipliers (dual variables) associated with rows (18) of (R_k). Then the reduced cost, or price, $P_k(j)$ associated with increasing $x_k(j)$ from $L_k(j)$ is given by (19).

$$(19) \qquad P_k(j) = -\sum_{s=1}^{m_{k+1}} \pi(s) B_k(s,j)$$

Clearly this price $P_k(j)$ is not a valid bound on the minimum change in objective value for (R_k) given an increase in $x_k(j)$; the other period k variables have been artificially forced to their lower bounds in order to compute $P_k(j)$. Therefore these prices cannot be used, as penalties are, to speed fathoming and ascertain which branches are <u>forced</u> (a branch is forced if condition (20) holds, where PEN can be either an up or down penalty on $x_k(j)$ at node i).

$$\text{(20)} \qquad \text{cumobj} + \text{FLOOR}(z_k(i) + \text{PEN}(j)) + \text{maxc}(k) \leq \text{incval} \quad ^*$$

The prices $P_k(j)$ <u>are</u> effective as measures of the general nature of the future impact of increasing $x_k(j)$, especially when they are used for comparison purposes only. When comparing $P_k(j)$ with $P_k(j')$, we are in some sense judging the relative value to future periods of one unit of activity $x_k(j)$ and one unit of activity $x_k(j')$.

Alternative means of estimating the impact of a change in $x_k(j)$ on the remaining periods do exist. For example, the single constraint $x_k(j) = L_k(j)$ could be appended to (R_k) and a reduced cost computed after solving this problem. However it is not clear that these prices would necessarily be more effective in guiding the subproblem search. Furthermore, a significant amount of extra computational work would be required. The prices $P_k(j)$ can be easily calculated at the same time as the bound maxc(k), as we shall see in Chapter 4. In fact, $P_k(j)$ can be obtained at essentially no additional cost in computer time and no increase in program complexity. Therefore, if these prices lead to a reasonable search strategy, they are preferable to the many other possibilities.

*As for the fathoming criterion, (Step 4), we do not round off if the costs are not all integral.

It should be noted that the prices $P_k(j)$ are more likely to give an accurate indication of the nature of the true future penalty functions if $U_k(j) = L_k(j) + 1$, as in a binary integer program. To understand this, consider a problem for which $U_k(j) = 10$, $L_k(j) = 0$. At various times during execution, branch decisions may have to be made for $x_k(j)$. Clearly $P_k(j)$ is a better criterion for judging the cost of increasing $x_k(j)$ from 0 to 1 than from 9 to 10, or even from 1 to 2. Unfortunately, it is not computationally feasible to calculate prices for all levels of each variable, particularly if the values of the other variables are not held fixed at their lower bounds. For this reason, $P_k(j)$ is used for guiding the search in spite of the drawback mentioned above. Substantial improvement in search direction would be expected if some more dynamic method of calculating future penalties were implemented (see the discussion on pseudocosts in Chapter 7).

Only computational experience can give a true evaluation of the effectiveness of the look-ahead prices $P_k(j)$. The results reported in Chapter 6 were obtained by directly adding the price $P_k(j)$ to the up penalty for variable $x_k(j)$ after fathoming and forced branching tests are performed. Thus these prices are used only to help guide the branch-and-bound search, so that feasible solutions to S_k which are more likely to yield improved complete solutions are examined first. Empirical evidence suggests that for most classes of test problems, the look-ahead prices do indeed lead to good search directions.

4. Algorithmic Modifications for Some Special Cases

The version of the SDA described in Section 2 was designed to find an optimal solution to the most general staircase ILP formulation. Often additional information on the structure or characteristics of the constraint matrix A is available. There are a few simple ways to alter the SDA to take advantage of two of these special cases.

One special case occurs when all coefficients of the constraint matrix are nonnegative. To take advantage of this property, the right-hand side $\hat{b}_t$ of S_t must be updated and tested after each branch on a period $t-1$ variable (normally $\hat{b}_t$ is calculated after a forward step). Since $B_{t-1} \geq 0$, we can generate the bound (21) on the change $-B_{t-1}x_{t-1}$ in the right-hand side of S_t, where $\tilde{x}_{t-1}(j)$ is the lower bound for $x_{t-1}(j)$ at the current node.*

$$-B_{t-1}x_{t-1} \leq -B_{t-1}\tilde{x}_{t-1} \tag{21}$$

Without loss of generality, assume that the problem has been transformed so that $L_t = 0$ for $t = 1, \ldots, T$. Then an extra fathoming condition, given by (22), can be tested in Step 4 of the SDA.

$$b_t - B_{t-1}\tilde{x}_{t-1} \geq 0 \tag{22}$$

If (22) is violated, $S_t(x_{t-1})$ will be infeasible, since both $A_t, x_t \geq 0$. Thus we can immediately fathom the current node; at this point it is not necessary to take a forward step to the next subproblem

*Recall that each branch to a new node resets the variable bounds.

(which would entail branching on all the period t variables not currently fixed at some integral value) to determine its infeasibility. This increased fathoming power (at minimal computational expense) should enhance the efficiency of the SDA for this special case.

The staircase constraint matrix A may have a special type of structure in which many of the variables are local to one period. In this case, the matrix A takes on the form exemplified in Figure 6.

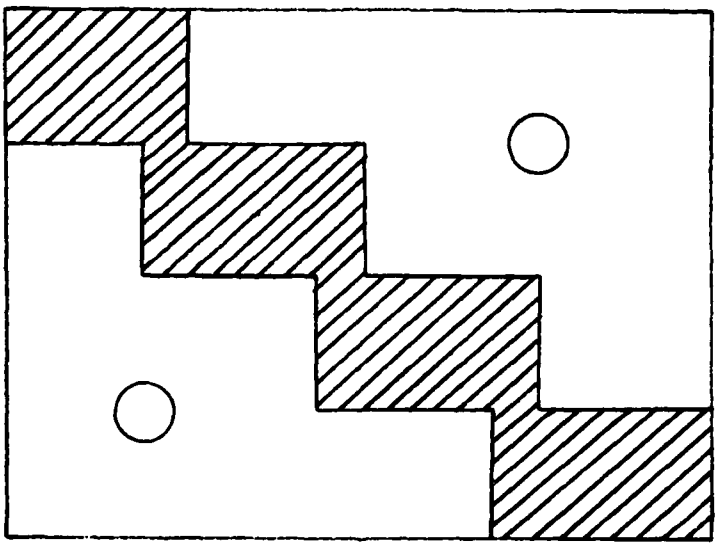


Figure 6: Staircase Matrix with Non-Linking Columns

As originally suggested by Schrage [1972], problems with some linking and some non-linking columns can be solved most efficiently by branching first on the linking variables. This idea can be applied to the subproblem branch-and-bound scheme (Step 4) of the SDA. After all the linking variables j of period t have been fixed at integral values $x_t(j)$, the remaining problem is truly independent of the other

subproblems. Thus the local part of subproblem t can immediately be solved to optimality; it will not be necessary to examine every feasible local solution.

Alternatively, a solution to the local part of S_t need not be calculated right away. If an upper bound on the optimal solution to this local problem (such as the optimum value of its LP relaxation) is generated, the SDA can take a forward step to the next time period. The algorithm would return to find the optimal values for the local variables only after finding a promising set of values for the linking variables in all time periods. This additional decomposition nested within the staircase decomposition would allow the SDA to disregard the local variables (as far as branching on them is concerned) until the problem is truly decomposed into T independent periods. As demonstrated by Reardon [1974] on dual angular problems, such a branching strategy leads to considerable improvement in execution times.

5. Advantages and Disadvantages of the SDA

Is the SDA a "good" algorithm? The best measure of the merit of a solution procedure is how well it performs in practice. The analysis of computational experience in Chapter 6 shows that decomposing the problem does usually lead to a substantial decrease in execution times. However it is also important to understand why this method succeeds, and what its weaknesses are, so that we may continue to improve it.

The SDA possesses all of the intrinsic advantages of decomposition methods. The branch-and-bound search, Step 4, is carried out on small subproblems, each with approximately n/t variables and m/t rows. These subproblems can generally be solved entirely in core, i.e., without having to access external storage devices such as disks, tapes, and drums. In contrast, valuable computer time may be squandered by excessive paging while solving the full problem, especially in a time-sharing environment.

The reduced size of the subproblems also considerably accelerates the execution of the linear programming segments of the algorithm. In fact, the solution times for the LP routines decrease faster than linearly with the number of subproblems, because the simplex method solves a typical practical problem in time roughly proportional to the cube of the number of rows. This is a crucial observation, since a large percentage of the total execution time of the LP-based branch-and-bound method for solving ILPs is spent in the dual simplex reoptimization phase. The speed of this part of the procedure enables the SDA to branch and to perform the reoptimization after a forward step very efficiently.

An added bonus of the reduction in problem size is that the small problems tend to be more stable numerically. This means that basis reinversion can be done less frequently.

One of the most important properties of the SDA is its ability to quickly locate good, feasible solutions. The two main reasons that it is able to accomplish this are the rapidity with which feasible subproblem solutions can be found and the excellent search directions normally produced by the combination of look-ahead prices and Tomlin's

improved penalties. The advantage of finding feasible, near-optimum integer points early is that for very large problems, the cost of solution to optimality may be prohibitive. Premature termination may be the only option available in this case. Computational results indicate that early termination of the SDA quite often yields a better solution than comparable termination of a standard branch-and-bound code. In addition, the SDA usually finds its first feasible solution within a few seconds (for moderate-sized problems). It is therefore likely that the SDA would be a good choice as an heuristic to generate starting solutions for methods such as Hillier's accelerated bound-and-scan algorithm. Further modifications along heuristic lines are discussed in Chapter 7.

One disadvantage of the SDA in comparison to a standard branch-and-bound approach is that the branching order is partially restricted; all variables in time period t must be set at fixed values before the algorithm can consider branching on period $t + 1$ variables. If it should happen that some variables in the last few periods are very important, then a branch-and-bound search working on the undecomposed problem might prove superior by virtue of the ability to branch on these crucial variables first. Fortunately, problems arising from practical multitime period models generally discount costs over time. In the discounted case, variables in early periods should be given priority over those in later periods. Therefore the lack of full choice in branching strategy should not be a serious hindrance to expeditious solution of the problem.

Another problem associated with the SDA is that it has less fathoming power than a standard LP-based branch-and-bound code. The SDA can not fathom a node simply because the optimum LP solution to

the current subproblem is integral; other feasible integral points may prove globally superior to this locally optimal (optimal for the current period) solution, so they must be considered. Furthermore, the standard code has the advantage of full information at each stage. This complete information allows it to calculate the strongest possible conditional bounds for fathoming purposes. The SDA must make do with knowledge of the current and previous subproblems, and only partial information about the future (in the form of look-ahead prices and the objective value bound maxc(k)). In order to alleviate this difficulty as much as possible, two sets of objective value bounds are used for fathoming in the SDA (see Chapter 4).

Less fathoming power implies more branches must be taken. In fact, the SDA typically (though not always) examines three to five times as many nodes as an algorithm working on the undecomposed problem. Nevertheless, the SDA will be faster on most problems because each branch can be taken much more efficiently. This is the key to the success of the SDA.

CHAPTER 4: OBJECTIVE BOUNDS FOR FATHOMING

1. A Bound on the Future Objective Value

The efficiency of the algorithm described in the preceding chapter is heavily dependent on its ability to fathom partial solutions before they are completed, preferably as early as possible. In the absence of a good objective bound on an optimal completion of the current partial solution, the SDA would be forced to find completions for most of the feasible solutions to the current subproblem. The set of complete solutions examined could quickly grow to an unmanageable size, and the performance of the algorithm suffer correspondingly.

In order to develop a bound tighter than the trivial bound (12) given in Chapter 2, consider the problem (R_k) defined in Section 3 of the third chapter and repeated here for convenience.

$$\text{maximize} \quad \sum_{t=k+1}^{T} c_t x_t$$

$$(R_k) \quad \text{subject to} \quad B_{t-1}x_{t-1} + A_t x_t \leq b_t \qquad t = k+1, \ldots, T$$

$$L_t(j) \leq x_t(j) \leq U_t(j), \quad j = 1, \ldots, n_t, \; t = k, \ldots, T.$$

Problem (R_k) is the LP relaxation of the aggregation of subproblems $k+1, \ldots, T$. From the perspective of time period k, the optimal solution to (R_k) represents the best that can be accomplished in the future, disregarding the past and the present.

Let maxc(k) be the optimal objective value of (R_k), for $k = 1, \ldots, T-1$. Define $\text{maxc}(T) = 0$. Furthermore, let z_k be the optimal objective value of subproblem $S_k(x_{k-1})$. Armed with these concepts, we can prove the following theorem.

Theorem: Let $x = (x_1, x_2, \ldots, x_{k-1})$ be any partial solution of (P), where $1 \leq k \leq T$. Then for every completion x' of x, the objective bound (23) is valid*.

$$cx' \leq \sum_{t=1}^{k-1} c_t x_t + z_k + \text{maxc}(k) \tag{23}$$

Proof:

For $k = T$, (23) is merely a restatement of the optimality of the objective value z_T. That is, (23) is reduced to $c_T x'_T \leq z_T$ for all x' feasible for S_T. This is true by definition of z_T.

Consider now $k < T$. Since x is a partial solution of (P), x_t is feasible for S_t for $t \leq k-1$. Moreover, the completion x' must be such that its components x'_t satisfy the constraints of $S_t(x'_{t-1})$, for $k \leq t \leq T$. Thus $(y, w_k) = (x', x'_k)$ is also feasible for the separable programming problem (24) given below.

*For $k = 1$, x is the null vector and the sum in (23) is defined to be 0.

$$\text{maximize} \quad \sum_{t=1}^{T} c_t y_t$$

$$\text{subject to} \quad y_t = x_t, \quad t = 1, \ldots, k-1$$

$$B_{k-1}x_{k-1} + A_k y_k \leq b_k$$

(24)
$$B_k w_k + A_{k+1} y_{k+1} \leq b_{k+1}$$

$$B_{t-1}y_{t-1} + A_t y_t \leq b_t, \quad t = k+2, \ldots, T$$

$$L_t(j) \leq y_t(j) \leq U_t(j), \; j = 1, \ldots, n_t, \quad t = 1, \ldots, T$$

$$L_k(j) \leq w_k(j) \leq U_k(j), \; j = 1, \ldots, n_k$$

The variable w_k, which does not appear in the objective function, has been introduced to make (24) separable into 3 independent problems. To solve (24) for an optimum value y^o, the first $k-1$ variables are fixed at predetermined values x_t. The variable y_k is determined by solving the LP relaxation of $S_k(x_{k-1})$, and $y_{k+1}, \ldots, y_T$, and w_k are found by solving (R_k). Let (y^o, w^o) be an optimum for (24). Then since (x', x_k') is feasible for this maximization problem,

$$cx' \leq cy^o = \sum_{t=1}^{k-1} c_t x_t + z_k + \text{maxc}(k) \qquad \boxtimes$$

Obviously, if the right-hand side of (23) is less than the incumbent objective value $c\bar{x}$, no completion of x can lead to an improvement in the objective function. Therefore, there is implicit in (23) a fathoming criterion for the SDA.

As the branch-and-bound search of subproblem S_k proceeds, the bound associated with (23) should be strengthened to reflect the extra information at the current node i. In particular, (23) is modified by replacing z_k with $z_k(i) + GP_k(i)$, where (as before) $GP_k(i)$ is the maximum Gomory penalty associated with node i of subproblem k, and $z_k(i)$ is the objective value of the LP relaxation of node i. This alteration yields the bound (25), which is equivalent to fathoming criterion (b) of Step 4 of the SDA (see Chapter 3).

$$(25) \qquad \sum_{t=1}^{k-1} c_t x_t + z_k(i) + GP_k(i) + \text{maxc}(k) \leq c\bar{x}$$

The problem (R_k) is independent of any branching decisions or search strategy implemented by the SDA. For this reason, it can be solved exactly once, as part of the initialization of the SDA. To find the values $\text{maxc}(k)$, $k = 1, \ldots, T$, efficiently, the following iterative procedure is executed in Step 2 of the SDA (see Figure 5).

Calculation of Bounds Maxc(k) and Prices $P_k(j)$, for all k

1. Set $\text{maxc}(T) = 0$ and $k = T - 1$. Input bounds on all the variables. (R_T) has no other constraints.
2. Adjoin the period $k + 1$ constraints to (R_{k+1}).

3. Let the objective function include the term $c_{k+1}x_{k+1}$. This completes the transformation of (R_{k+1}) into (R_k).
4. Temporarily set $x_k = L_k$, and solve for an optimum set of dual variables. Use these simplex multipliers to calculate $P_k(j)$, the look-ahead prices (see Chapter 3).
5. Relax the requirement $x_k = L_k$, and solve (R_k). The optimal objective value is maxc(k).
6. Set $k = k - 1$. If $k = 0$, stop. Otherwise, go to 2.

Note that the prices $P_k(j)$ are computed at relatively little cost. Also, by adjoining constraints to an LP which has already been solved, we are likely to have a good starting solution for the simplex method. In practice, execution of the procedure described above usually comprises a small percentage of total solution time.

Values for $z_k(i)$ and $GP_k(i)$ are available as a by-product of the branch-and-bound search of S_k. The sum in (25) can be stored as a variable called cumobj (cumulative objective value), and updated with each forward and backward step. Therefore, since the bound maxc(k) can be computed initially, the iterative portion of the SDA need never involve the examination of data from more than one subproblem at a time. This property was required to make the SDA a true decomposition algorithm.

2. A Second Type of Bounding Procedure

The bound $\text{maxc}(k)$ defined in the preceding section provides a valid bound on the objective value of the future, as portrayed by (R_k). However, by assigning a cost of 0 to the period k variables appearing in (R_k), unrealistic values of these variables may occur in an optimum solution. In an attempt to correct this fault, another set of bounds, called $\text{maxc}(\lambda,k)$, has been developed.

One way to generate more realistic values for x_k would be to introduce the term $c_k x_k$ into the objective function of (R_k). For $c_k > 0$, this modification would cause an increase in at least some of the variables $x_k(j)$ in an optimal solution to (R_k). It is probable that $x_k(j)$ would in fact rise to levels somewhat higher than those at which they appear in an optimal solution to the LP relaxation of (P). This inflation of the period k variables is attributable to the fact that (R_k) is a relaxation of (P); the period k constraints are not present in the formulation of (R_k).

There is a problem with adding $c_k x_k$ directly into the objective function of (R_k). If this were done, the contribution of the period k variables would be counted twice in (25). To avoid this double-counting, define $\underline{\text{maxc}(\lambda,k)}$ to be the optimal objective value of problem (Q_k) stated below, for some fixed $\lambda \in [0,1]$.

$$\text{maximize} \quad \sum_{t=k+1}^{T} c_t x_t + \lambda(c_k x_k)$$

$$(Q_k) \quad \text{subject to} \quad B_{t-1}x_{t-1} + A_t x_t \leq b_t, \qquad t = k+1, \ldots, T$$

$$L_t(j) \leq x_t(j) \leq U_t(j), \quad j = 1, \ldots, n_t, \quad t = k, \ldots, T.$$

This definition of $\text{maxc}(\lambda,k)$ holds for $k = 1, \ldots, T - 1$. For $k = T$, no bounds are needed on the future (since our finite horizon problem ends at period T). In this case, (23) is as tight a bound as is available for the SDA.

As depicted in Figure 7, the constraints of problem (Q_k) are identical to those of (R_k), and the objective function differs only in the term $\lambda(c_k x_k)$. Ideally, the parameter λ should reflect the influence of the period $k + 1$ constraints in determining x_k. In practice, the weights $(1 - \lambda)$, λ that should be assigned to the period k and $k + 1$ constraints vary considerably with the settings of the variables of periods $1, \ldots, k - 1$.

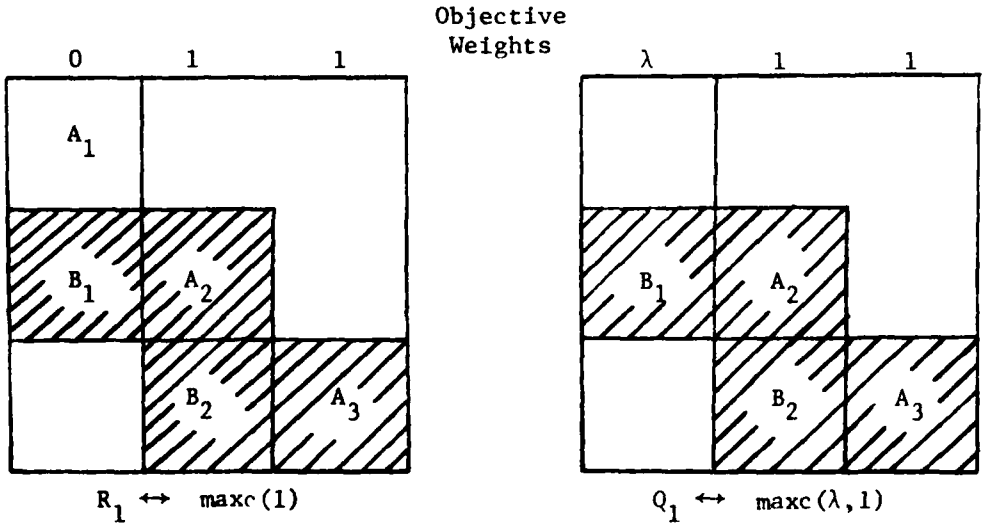


Figure 7: Correspondence Between R_1 and Q_1

The definition of $\text{maxc}(\lambda,k)$ given above leads to the fathoming criterion (26).

$$\sum_{t=1}^{k-1} c_t x_t + (1 - \lambda) z_k + \text{maxc}(\lambda,k) \leq c\bar{x} \tag{26}$$

Here, as in the preceding section, z_k can be replaced by $z_k(i) + GP_k(i)$ as the branch-and-bound search produces more information about the period k subproblem.

To verify that (26) is a valid fathoming criterion, we have a generalization of the theorem in Section 1.

Theorem: Let $x = (x_1, x_2, \ldots, x_{k-1})$ be any partial solution of (P), where $1 \leq k \leq T - 1$. Then for every completion x' of x, the objective bound (27) is valid.

$$cx' \leq \sum_{t=1}^{k-1} c_t x_t + (1 - \lambda) z_k + \text{maxc}(\lambda,k) \tag{27}$$

Proof:

Since x' is a complete solution of (P), we have that x_1' is feasible for S_1 and x_t' is feasible for $S_t(x_{t-1}')$, $2 \leq t \leq T$. Therefore, $(y,w_k) = (x', x_k')$ satisfies the constraints of the separable programming problem (28), where $c_t' = c_t$ for $t \neq k$, and $c_k' = (1 - \lambda)c_k$.

$$\text{maximize} \quad \sum_{t=1}^{T} c_t' y_t + \lambda c_k w_k$$

$$\text{subject to} \quad y_t = x_t, \qquad t = 1, \ldots, k-1$$

$$B_{k-1} x_{k-1} + A_k y_k \leq b_k$$

$$B_k w_k + A_{k+1} y_{k+1} \leq b_{k+1} \tag{28}$$

$$B_{t-1} y_{t-1} + A_t y_t \leq b_t, \qquad t = k+2, \ldots, T$$

$$L_t(j) \leq y_t(j) \leq U_t(j), \quad j = 1, \ldots, n_t, \; t = 1, \ldots, T$$

$$L_k(j) \leq w_k(j) \leq U_k(j), \quad j = 1, \ldots, n_k$$

The variable w_k has been introduced to relax the implicit constraint that y_k take on the same values in both the period k and $k+1$ constraints. This procedure is equivalent to a Lagrangean relaxation of (P) (see Geoffrion [1974]).

To solve (28) for an optimal solution (y^o, w^o), three independent problems must be solved. To find y_t^o, for $t < k$, solve the trivial problem defined by the equality constraints in (28); its solution is $y_t = x_t$, for $t = 1, 2, \ldots, k-1$. Secondly, the values of y_k^o can be found by solving $S_k(x_{k-1})$. Note that the objective contribution is $(1 - \lambda)$ times that of S_k. Finally, by solving (Q_k), an optimal set of values for $y_{k+1}, \ldots, y_T$, and w_k can be determined.

The fact that (x', x_k') is feasible for (28), together with the equation $cx' = c'x' + \lambda c_k x_k'$, implies that:

$$cx' \leq c'y^o + \lambda c_k w_k^o = \sum_{t=1}^{k-1} c_t x_t + (1 - \lambda) z_k + \text{maxc}(\lambda,k) \qquad \boxtimes$$

The bound $\text{maxc}(\lambda,k)$ can be computed in the initialization phase (Step 2 of the SDA) along with $\text{maxc}(k)$ and $P_k(j)$. The procedure outlined in Section 1 need only be altered so that the last step (6) is expanded into steps 6' and 7' given below.

6'. Now that (R_k) has been solved, change the objective function by including the term $\lambda(c_k x_k)$ to create (Q_k). Reoptimize via the primal simplex method; set $\text{maxc}(\lambda,k)$ equal to the optimal objective value of (Q_k).

7'. Set $k = k - 1$. If $k = 0$, stop. Otherwise go to 2.

Notice that the solution of (R_k) makes available a good, feasible starting point with which the simplex method can begin to solve (Q_k) in step 6'. Thus $\text{maxc}(\lambda,k)$ can be obtained at relatively little extra computational expense after finding $\text{maxc}(k)$ -- which is, in fact, equal to $\text{maxc}(0,k)$.

Similarly, for any set of values $\{\lambda_i, i = 1, \ldots, r\}$, only a series of primal simplex reoptimizations is necessary to determine the bounds $\text{maxc}(\lambda_i,k)$ once $\text{maxc}(k)$ has been calculated. The "best" choice of r (the number of bounds generated) and the values of the parameters λ_i is highly problem dependent. Furthermore, the strength of any particular bound changes dramatically during the course of the solution procedure, because the variables of period $k - 1$ become fixed at widely varying values (hence the right-hand side of the period k

constraints also fluctuates substantially). Finding criteria for making decisions on the values r and $\{\lambda_i\}$ is an interesting question for future research.

One easy result can be obtained for the case in which it is desirable to find only one λ, especially when $k = 1$.

Theorem: For $k = 1$, a value λ^* which gives the tightest possible bound in (27) can be found by solving a linear program.

Proof:

The parameter λ^* can be found by solving the quadratic program (29).

$$
\begin{aligned}
&\text{minimize} \quad \{\text{maximum} \quad (1-\lambda)c_1x_1 + \lambda c_1y_1 + \sum_{t=2}^{T} c_t x_t\} \\
&0 \le \lambda \le 1 \\
&\qquad\qquad \text{subject to} \quad B_{t-1}x_{t-1} + A_t x_t \le b_t \qquad t = 1, 3, 4, \ldots, T \\
&\qquad\qquad\qquad\qquad\qquad\qquad B_1 y_1 + A_2 x_2 \le b_2 \qquad (29) \\
&\qquad\qquad\qquad\qquad\qquad\qquad L_t(j) \le x_t(j) \le U_t(j), \quad j = 1, \ldots, n_t, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad t = 1, \ldots, T. \\
&\qquad\qquad\qquad\qquad\qquad\qquad L_1(j) \le y_1(j) \le U_1(j), \quad j = 1, \ldots, n_1
\end{aligned}
$$

In (29), the implicit constraint $y_1 = x_1$ has been dropped, as in a Lagrangean relaxation of (P).

By taking the dual of the inner maximization problem, the quadratic terms $(1 - \lambda)c_1x_1$ and λc_1y_1 are eliminated. In this dual formulation, which is a minimization problem, the parameter λ does not appear in the objective function; it is found only on the right-hand side of the constraints. The desired linear programming problem emerges when these terms are moved to the other side of the inequalities and the inner and outer minimizations are combined. ⊠

A similar result can be obtained for any period k, provided the values of the past variables (variables from periods $1, \ldots, t - 1$) are close to those of the optimum x^o of the LP relaxation of (P). Unfortunately, this condition is satisfied only a small percentage of the time during the course of the algorithm.

3. Computational Comparison of Bounding Procedures

The value of λ which yields the tightest bound (27) changes as different partial solutions are examined. For some values of x_{k-1}, the objective value of $S_k(x_{k-1})$ will be relatively large; the most effective bound in this case will have a value of λ closer to 1 than to 0. On the other hand, a greedy setting for x_{k-1} might result in a small right-hand side for subproblem k. In this case, a small value of λ will maximize the right-hand side of (27).

No single value of λ can dominate all other possibilities throughout the entire branch-and-bound search. However, it may be that, in practice, maxc(k) produces ineffective bounds compared to maxc(λ,k),

or vice versa. In order to further explore this question, the bounds created from maxc(k) and maxc($\frac{1}{2}$,k) were tested for effectiveness, both individually and together, on several moderate-sized problems. The choice of $\lambda = \frac{1}{2}$ was made in an attempt to strike a balance between neglecting the influence of the objective contribution of x_k (by giving it a weight of 0 in (Q_k)) and overemphasizing it (a weight near 1 would be too large in light of the fact that some constraints affecting x_k do not appear in (Q_k)).

A representative sample of the results of testing these two objective bounds is given below in Table 1. The number of branches taken is a reliable criterion for judging the fathoming power of a particular objective bound since solution time is directly proportional to this number. On some of the smaller problems, the number of branches taken using only the trivial bound (12) for fathoming is included for comparison purposes. This information is contained in the category marked "neither". The column designated "both" exhibits the number of branches taken when a node is considered fathomed if either (25) or (26) (with $\lambda = \frac{1}{2}$) is satisfied.

Note that maxc(k) provides a more effective fathoming criterion for some problems, while maxc($\frac{1}{2}$,k) is more effective for others. Either bound alone is far superior to the trivial bound (12), but there is also a marked improvement in all cases when both bounds are used. Because of these findings, the version of the SDA developed for testing (see Chapter 6) uses both maxc(k) and maxc($\frac{1}{2}$,k) to produce objective bounds for fathoming.

PROBLEM SIZE			TOTAL NUMBER OF BRANCHES TAKEN			
rows	vars	periods	both	maxc(k)	maxc($\frac{1}{2}$,k)	neither
20	20	4	1004	1164	1076	2,083
20	32	4	1779	2193	4117	28,055
20	40	4	1942	2646	3359	>100,000
25	45	5	3336	4272	3872	
40	40	5	1656	2072	2165	
30	50	5	8502	20280	11997	
30	50	5	44804	64311	66614	

Table 1: Comparison of Bounding Procedures

CHAPTER 5: EXTENSIONS OF THE STAIRCASE ALGORITHM

1. Higher Order Staircase Constraint Matrices

The concept of a multitime period structure can be generalized to include constraints which link nonadjacent time periods. For example, if the period t constraints have nonzero coefficients only for variables of periods t, $t - 1$, and $t - 2$, then the matrix is said to be a staircase matrix of order 2 (see Figure 8). Similarly, the rows of an r^{th} order staircase matrix may link activities from as many as $r + 1$ consecutive periods. Applications of models with higher order staircase structures arise in industries for which construction of new plants or capacity expansion of existing facilities requires several time periods to complete. The power industry exhibits this kind of behavior due to long lead times for plant construction and further delays necessary for satisfaction of environmental standards.

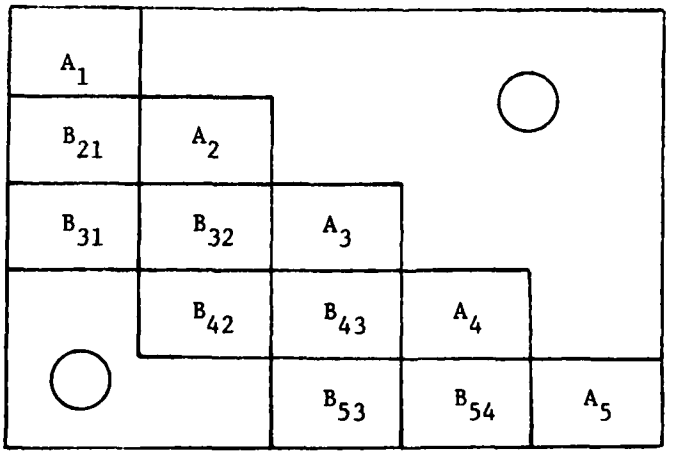


Figure 8. Staircase Matrix of Order 2

The order of a staircase matrix can be reduced by the addition of new constraints and variables. For each variable $x_t(j)$ which links time periods t and $t + r$, a new variable y can be introduced into period $t' = t + [r/2]$. Replacing $x_t(j)$ by y in the constraints of periods t', ..., $t + r$ eliminates the long interperiod dependencies due to $x_t(j)$. By equating these two variables (via the period t' constraint $x_t(j) = y$), the original problem is unchanged, although the structure has been simplified. This process reduces the order of the staircase matrix by at least 1 (by $[r/2]$ if all variables are treated in this way).

With the addition of enough extra constraints and variables, any r^{th} order staircase matrix can be transformed into an equivalent staircase matrix (of order 1). The resulting staircase problem could be solved by the SDA; the optimal values of the original variables $x_t(j)$ would be the optimal values for the original problem as well. Although the increase in problem size renders this approach impractical for matrices with many linking variables, it is probably the best method of attack when only a few columns violate the staircase structure, as in Figure 9.

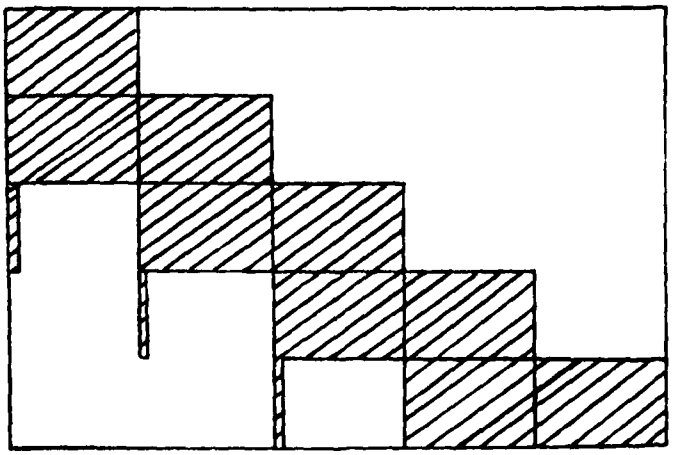


Figure 9: "Almost" Staircase Matrix

Alternatively, problems with only a few variables that link nonadjacent periods could be solved by modifying the staircase algorithm rather than the constraint matrix. Those few variables could be given priority; once they are fixed at integral values the problem is truly staircase in structure. However it is not even necessary to force the initial branches to be taken on these variables. The decomposition approach of the SDA ensures that they will be fixed before any future period is examined. Some simple changes to the bounding and pricing routines are sufficient to account for the effect of these variables on future periods. With these alterations, the SDA can be used to solve a problem with the structure shown in Figure 9.

The ideas expressed in the preceding paragraph can be extended to all higher order staircase problems. Consider a multitime period model for which many variables link three consecutive periods (i.e., an order 2 staircase problem) as depicted in Figure 8. Proceed as in the original version of the SDA: variables in each period are fixed before moving on to the next period. After setting the variables from periods $t - 2$ and $t - 1$ to fixed integer values, the only variables which influence the period t constraints are those from period t. Therefore the subproblems can again be solved by looking only at the small diagonal blocks. The sole change from the original formulation used by the SDA is that the right-hand side of each subproblem S_t depends on values passed forward from both periods $t - 1$ and $t - 2$ (transformed by the offdiagonal submatrices $B_{t,t-1}$ and $B_{t,t-2}$). The subproblems corresponding to this structure have the following form:

$$\text{maximize} \quad c_t x_t$$

$$\text{subject to} \quad A_t x_t \leq b_t - B_{t,t-1} x_{t-1} - B_{t,t-2} x_{t-2}$$

(S_t)

$$L_t \leq x_t \leq U_t$$

$$x_t \text{ integer}$$

For the general r^{th} order staircase problem, the right-hand side of subproblem t will have terms depending on the (fixed) values of the period k variables, for $t - r \leq k \leq t - 1$. The overall structure of the algorithm (as given in Chapter 3) does not change when it is applied to this problem, although some of the procedures must be modified slightly.

The look-ahead prices $P_k(j)$ described in Chapter 3 can be calculated in exactly the same fashion for the higher order staircase problem. The aggregation of future subproblems (R_k) inherits the r^{th} order structure from the original problem, but otherwise the computation is unchanged. As before, these prices can be used to help guide the branch-and-bound search for feasible subproblem solutions.

The bounds for fathoming developed in Chapter 4 must be adjusted to reflect the presence of variables from several "past" periods in the problem (Q_k). In fact, it would probably be advantageous to assign different weights λ_i to the objective function contributions of the period i variables, for $i \leq k$, appearing in (Q_k). For example, in the case $r = 2$, it might be desirable (for reasons of symmetry) to choose $\lambda_{k-1} = \frac{1}{3}$, $\lambda_k = \frac{2}{3}$.

The fathoming criterion in the general case becomes:

$$(30) \qquad \sum_{t=1}^{k-1} (1 - \lambda_t)c_t x_t + (1 - \lambda_k)z_k + \text{maxc}(\lambda,k) \leq c\bar{x}$$

where $\lambda_t = 0$ for $t \leq k - r$ and $\text{maxc}(\lambda,k)$ is the optimal objective value of the problem (Q_k) given below.

$$\text{maximize} \qquad \sum_{t=k-r+1}^{k} \lambda_t c_t x_t + \sum_{t=k+1}^{T} c_t x_t$$

$$(Q_k) \quad \text{subject to} \quad \sum_{i=1}^{r} B_{t,t-i} x_{t-i} + A_t x_t \leq b_t, \text{ for } t = k + 1, \ldots, T$$

$$L_t \leq x_t \leq U_t, \text{ for } t = 1, \ldots, T$$

A staircase matrix of order $T - 1$ is also called a lower block triangular matrix (see Chapter 1 and Figure 2). The ability of the SDA (with the modifications detailed above) to solve a problem with this structure is important because any linear set of constraints can be transformed into lower block triangular form. In fact, a rearrangement of the rows and columns of a typical (somewhat sparse) constraint matrix is often sufficient to put the problem in this form (see, e.g., Weil and Kettler [1971] for block-angular rearrangement methods). Thus if the SDA performed well on lower block triangular matrices, it could be a good algorithm for the general, unstructured integer programming problem.

In order to test this possibility, a version of the SDA was designed to solve integer programming problems with lower block triangular structure. Unfortunately, computational tests (using $\lambda_t = 0$ and $\frac{1}{2}$) revealed that the decomposition approach is not very efficient for solving randomly generated problems of this type (see Chapter 6 for a description of the methods employed to generate test problems). Apparently too much useful information is lost by examining only the small subproblems. In particular, the look-ahead prices are inadequate for guiding the branch-and-bound search. The fathoming criterion (30) is also not as effective as objective function bounds which can be generated if information from the undecomposed problem is available.

To find a better global search strategy, pseudocosts could be employed (see Chapter 7). Although the use of pseudocosts should eliminate the problem of poor search directions, fathoming would still be a major problem. The large variations in the right-hand side of each subproblem which accumulate as the variables of the past are fixed cause the solution of problem (Q_k) to bear less and less relation to actual LP-optimal settings of the variables. For fixed values of λ_t, the bound (30) does not seem to be useful often enough to make the decomposition approach competitive with a standard branch-and-bound solution technique applied to the undecomposed problem. Thus it appears that the decomposition method is most successful when applied to problems with staircase structured matrices of low order.

2. Alternative Search Strategies

The LP-based branch-and-bound algorithm was chosen as the appropriate subproblem search procedure for reasons detailed in Chapter 3. Briefly, this technique has proven to be computationally fast (relative to other integer programming methods), is easy to reestablish after a change in the right-hand side, and provides flexible search strategies. Nevertheless, some of the other integer programming algorithms might search the subproblems more efficiently when the problem has special characteristics.

In particular, a greedy algorithm, such as the dynamic programming method of Cooper and Cooper (see Chapter 1), could work very well on problems with weak interperiod relationships. For such problems, locally optimal solutions at each stage should quickly lead to a good (although usually suboptimal) feasible solution. The advantage of the greedy algorithm is that it rapidly discovers several of the (locally) best solutions, although a complete search of the feasible solutions would be quite time-consuming. Thus fathoming capabilities might be improved by the exceptionally fast location of a good feasible integer point. It might also be possible to modify this greedy search strategy by a parametric change in the objective function corresponding to some measure of the effect of each variable on future subproblems. Provided that set-up time after changes in the right-hand side has a minimal impact on total solution time, this method could be quite successful.

Group theoretic methods could prove valuable in solving subproblems for which the determinant D of most dual feasible bases is small in magnitude. The procedure for solving the group theoretic

formulation (GTP) (see Chapter 1) would be easy to reestablish after a forward step, because the dual feasibility of a basis is unaffected by a change in the right-hand side. However the search for feasible subproblem solutions would require the solution of many shortest route problems, which increase in complexity with $|D|$. Thus this approach would be efficient only for small values of $|D|$.

In multitime period models, the diagonal submatrices often have a structure that is not necessarily shared by the matrix as a whole. For example, these smaller matrices might describe transportation or network flow problems. A solution technique designed to work with the entire undecomposed problem might not be able to take advantage of this additional structure, especially if the offdiagonal blocks do not have the same characteristics.

The decomposition technique described in Chapter 2 can capitalize on this embedded structure. The subproblems (S_t) are determined by the diagonal blocks; only the magnitude of the (constant) right-hand side is affected by the offdiagonal blocks. Therefore a specialized algorithm could be developed to expedite the search for feasible solutions to each subproblem by exploiting the special structure of its constraint matrix. This adaptability of the SDA should make it a very effective solution procedure for problems with special substructures.

3. Nonlinearity

In the preceding section, it was observed that the diagonal blocks may have a special structure that the decomposition algorithm can exploit. A method which does not decompose the problem must deal

with the characteristics of the matrix as a whole, including the off-diagonal blocks. For the SDA, the offdiagonal blocks are relatively unimportant once the initial stage of calculating look-ahead prices and bounds for fathoming has been completed. At this point, the algorithm can proceed in exactly the same manner whether or not the offdiagonal terms are linear.

To avoid difficulties with computation and interpretation of the look-ahead prices, pseudocosts should be used to guide the branch-and-bound search. With this substitution, the SDA can solve the following generalized integer programming problem (GIP):

$$
\begin{aligned}
&\text{maximize} && c_t x_t \\
&\text{subject to} && A_t x_t + f_t(x_{t-1}) \leq b_t, && t = 1, \ldots, T \\
&&& L_t \leq x_t \leq U_t, && t = 1, \ldots, T \\
&&& x_t \text{ integer}, && t = 1, \ldots, T
\end{aligned}
\tag{GIP}
$$

where $f_1(\cdot) \equiv 0$, and the only restrictions on the functions f_t are that the problems (Q_k), $k = 1, \ldots, T - 1$, are not too difficult to solve. As in Chapter 4, (Q_k) is obtained by aggregating the subproblems $S_{k+1}, \ldots, S_T$, and relaxing the integrality constraints (see below).

$$\text{maximize} \quad \lambda c_k x_k + \sum_{t=k+1}^{T} c_t x_t$$

$$(Q_k) \qquad \text{subject to} \quad A_t x_t + f_t(x_{t-1}) \leq b_t, \qquad t = k+1, \ldots, T$$

$$L_t \leq x_t \leq U_t, \qquad t = k, \ldots, T$$

The optimum objective value of (Q_k) can be used to develop the same fathoming criterion (26) given in Chapter 4. Since pseudocosts do not require any additional calculations on the undecomposed problem, the SDA can proceed as before except for a slight modification of the forward step. The change in the right-hand side of S_{k+1} is given now by the formula $\hat{b}_{k+1} = b_{k+1} - f_{k+1}(x_k)$. With this one alteration, the SDA can solve the problem (GIP). Moreover, it should have a clear advantage over any method which tries to solve the undecomposed problem, and hence must deal directly with the nonlinear constraints of (GIP).

In addition to nonlinear offdiagonal terms, nothing in the development of the SDA in Chapter 2 precludes a nonlinear objective function. Of course, the subproblem search procedure would have to use a nonlinear integer programming algorithm rather than the LP-based branch-and-bound search, but the basic structure of the decomposition method would not have to be altered. The same remark applies to nonlinearities in the diagonal blocks, although in this case the subproblems would inherit all the nonlinearities of the original system. The benefits of a decomposition approach in this instance would not be as clear as in the nonlinear offdiagonal (and linear diagonal) case.

4. Mixed Integer Linear Programs

A mixed integer linear program (MILP) consists of linear constraints and a linear objective function, plus integrality restrictions on a proper subset of the variables. If the constraint matrix (excluding the integrality constraints) has a staircase structure, the methods developed in the previous chapters can be applied to this problem. The staircase MILP can be formulated as follows:

$$\text{maximize} \quad \sum_{t=1}^{T} c_t x_t + \sum_{t=1}^{T} d_t y_t$$

$$\text{subject to} \quad A_t x_t + B_{t-1} x_{t-1} + D_t y_t + E_{t-1} y_{t-1} \leq b_t, \qquad t = 1, \ldots, T \tag{MILP}$$

$$L_t \leq x_t \leq U_t, \qquad t = 1, \ldots, T$$

$$x_t \text{ integer}, \quad t = 1, \ldots, T$$

In one special case, the SDA can solve a mixed integer program without any modifications whatsoever. Suppose the linking variables are all required to take on integral values, i.e., the continuous variables y_t are local. In terms of the formulation above, the matrices E_{t-1} are identically 0 for all t. For this problem, fixing all the integral variables of subproblem t, which has constraint matrix $(A_t \; D_t)$, leaves a linear programming problem in the local variables y_t. This local problem can be solved immediately for values of the continuous variables

before proceeding to the next period*. Since the continuous variables appear in precisely one subproblem, no complications ensue, and the decomposition algorithm encounters no difficulties in solving the problem.

In the more general case, for which at least some of the linking variables are continuous, difficulties do arise. After fixing the values of the integer variables of period 1 at, say, $x_1 = \alpha_1$, subproblem S_1 will have the following form:

$$\text{maximize} \quad c_1x_1 + d_1y_1$$

$$(S_1) \qquad \text{subject to} \quad A_1x_1 + D_1y_1 \leq b_1$$

$$x_1 = \alpha_1$$

Although this problem could be directly solved for values of the continuous variables y_1, it is improbable that those values would be optimal for the entire problem, even given the settings of the period 1 integer variables. The values of y_1 also affect the period 2 constraints, and this effect cannot be ignored. Unlike the integer variables, all the distinct possible values of y_1 cannot be explicitly examined, nor implicitly screened through a branch-and-bound type of search, because there are an infinite number of them. Therefore the constraints containing these variables must be carried forward into the next period. Subproblem S_2 thus becomes:

*In fact the solution will already have been calculated as a by-product of the branch-and-bound search.

$$\text{maximize} \quad c_2x_2 + d_2y_2 + d_1y_1$$

$$\text{subject to} \quad D_1y_1 \leq b_1 - A_1x_1$$

$$A_2x_2 + D_2y_2 + E_1y_1 \leq b_2 - B_1x_1 \qquad (S_2)$$

$$L_2 \leq x_2 \leq U_2, \; x_2 \text{ integer}$$

None of the continuous variables y_1 or y_2 can be set at any particular value, even after the period 2 integer variables are fixed, because of the connection through y_2 to subproblem 3. As the algorithm progresses towards the final period, constraints depending solely on the continuous variables of previous periods will continue to accumulate. Only when the last period is reached can definite values be placed on the continuous variables, as (S_T) (shown below) is solved to optimality.

$$\text{maximize} \quad c_Tx_T + \sum_{t=1}^{T} d_ty_t$$

$$\text{subject to} \quad D_1y_1 \leq b_1 - A_1x_1$$

$$D_ty_t + E_{t-1}y_{t-1} \leq b_t - A_tx_t - B_{t-1}x_{t-1}, \qquad (S_T)$$

$$\text{for} \quad t = 2, \ldots, T - 1$$

$$A_Tx_T + D_Ty_T + E_{T-1}y_{T-1} \leq b_T - B_{T-1}x_{T-1}$$

$$L_T \leq x_T \leq U_T, \; x_T \text{ integer}$$

Clearly, if the number of continuous variables is a substantial proportion of the total number of variables, this solution method will not be efficient. The advantages gained by decomposing the problem would be dissipated by the fact that later subproblems can have almost as many rows as the original problem. Since each subproblem must be resolved many times, this increase in the number of constraints will have a substantial adverse impact on solution times. Therefore, the decomposition approach is recommended for MILP only if the number of continuous variables is relatively small, or if none of the continuous variables link time periods.

What methods should be employed to solve a staircase MILP which has only a few integer variables and many (linking) continuous variables? The staircase decomposition algorithm does not adapt well to this problem, but there are other solution techniques available. The linear programming portions of a standard branch-and-bound algorithm can be modified to recognize and take advantage of the staircase structure. Techniques for simplifying the calculation and updating of the basis inverse, such as block triangularization (see Heesterman and Sandee [1965], Saigal [1966], Dantzig [1973], and Wolmer [1979]), could be used to accelerate both the primal simplex and dual simplex reoptimization phases of the algorithm. Alternatively, the sparse matrix and partial pricing methods developed for staircase LPs by Fourer [1980] could make the LP part of the algorithm extremely efficient.

Another possible approach is to use nested decomposition (see Chapter 1) to solve the LP relaxation of (MILP), followed by a branch-and-bound search over the feasible values of the integer variables.

Unfortunately, reoptimization after a branch would then be very difficult both because primal solutions are difficult to construct from the information present in the subproblems (even though optimal dual values are apparent) and because many proposals become infeasible after a branch. A more promising approach involves nested decomposition of the dual (Abrahamson [1980]). In this method, the dual linear program is decomposed into small subproblems, and information is passed both forward (in the form of "economic cuts") and backward ("feasibility cuts") between time periods until equilibrium, and hence optimality, is achieved. Branches in the primal problem correspond to adding or deepening dual economic cuts; reestablishing equilibrium after a change in one economic cut might not require too much extra work. Moreover, a solution to the primal is readily accessible. Therefore this staircase linear programming algorithm, although it is still in the development stage, could eventually provide a way to decompose and efficiently solve (MILP).

CHAPTER 6: COMPUTATIONAL RESULTS

1. Implementation and Data Generation

Evaluation of the staircase decomposition algorithm presented in this paper was carried out on the SCORE DEC System 20 computer at Stanford University. For this purpose, a computer code embodying all the features discussed in Chapters 3 and 4 was written in FORTRAN. Due to the restrictive nature of the FORTRAN-10 compiler at SCORE, few (if any) features of the current implementation would be unacceptable to a FORTRAN compiler on another system. The selection of FORTRAN as the programming language was not based on any natural inclinations of the author or characteristics of this language which make it a particularly good choice for encoding the SDA; it was selected to maintain compatibility with an existing integer programming code already written in FORTRAN.

The branch-and-bound search procedure, which is the heart of the algorithm, is a slightly modified version of the computer program BB written by Gary Kochman as part of his dissertation at Stanford University. This computer code solves pure integer programming problems with general upper and lower bounds on the variables using a branch-and-bound technique similar to that outlined in Chapter 1. Tomlin's [1971] improved penalties are employed to guide the choice of branching variable at unfathomed nodes, with the branch being taken in the direction opposite to the maximum penalty. Nodes are removed from the branch-and-bound list according to a last-in-first-out (LIFO) strategy, and reoptimization after a branch is accomplished by the dual simplex method. See Kochman [1976b] for further details about this program.

The linear programming portions of the code BB were developed by John Tomlin of the Systems Optimization Laboratory at Stanford University and adapted by Kochman to efficiently deal with simple upper bounds on the variables. The most important features of LPM-1 are: storage of the basis inverse in product form (see Orchard-Hays [1968]); LU decomposition of the basis inverse (see Benichou et al. [1977]); and storage of all data, including the inverse, in compressed form (i.e., zeroes are not saved). Because of these features, the program BB works efficiently on sparse matrices. This is crucial for solving a multitime period problem without using decomposition, since by definition a staircase matrix is quite sparse. In addition, if the diagonal blocks also happen to be sparse, these features of LPM-1 will help improve the performance of the SDA.

The test problems mentioned in the rest of this chapter were generated randomly using various parameters to influence the problem characteristics. Objective function coefficients were generated uniformly from the integers in the interval $[\ell,u]$; entries in the constraint matrix were determined in the same manner, using different parameters to describe the range of allowable values. For some problems the right-hand side was produced by the same process, but for others the right-hand side of the i^{th} constraint was generated deterministically from the formula (31), where the parameter τ is a measure of the tightness of the constraint.

$$b_i = \tau * \sum_{j=1}^{n} A(i,j)/n \qquad (31)$$

With this method of determining the right-hand sides, the tightness of the constraints is more uniform over the subproblems, making it easier to

judge the effects on solution times of varying problem dimensions. To complete the specification of the test problem, additional input parameters control the density of the diagonal and offdiagonal submatrices, the dimensions of each subproblem, the total number of periods, and the seed for the random number generator.

All solution times reported in this chapter are in CPU seconds, excluding output but including input times. These numbers should be used for comparison purposes only, as solution times might vary substantially on other computer systems, or if computer codes of professional caliber were substituted for BB and SDA. It should be noted however that any flaws inherent in the program BB are perforce repeated in the implementation of the SDA, which uses BB as a subroutine. Moreover the sparsity techniques present in LPM-1 ensure that any computational advantage enjoyed by the SDA over BB is truly a result of the decomposition, and not merely due to the fact that the SDA need not deal with the large number of zeroes present in the problem formulation. Therefore a direct comparison of solution times for SDA and BB (the latter on the undecomposed problem) is a fair test of the benefits of decomposing a staircase integer programming problem.

2. Empirical Success of the SDA

Several thousand test problems were generated and solved during the development of the staircase decomposition algorithm. Not all of these results will be presented and discussed here; however, representative examples have been selected to give an accurate picture of the overall effectiveness of the SDA. Special care was taken to ensure that these sample problems cover a broad range of possible data types.

Table 2 illustrates the behavior of the SDA on several small, relatively simple, binary test problems. The dimensions of each problem are given by m, the total number of constraints, n, the total number of binary variables, and T, the number of time periods. For these and similar examples, the SDA generally finds and verifies an optimal solution in approximately half the CPU time (given in seconds) necessary for the branch-and-bound algorithm BB. Both the rapid arrival at an optimal solution and the short total execution time are attributable to the greater speed in branching of the SDA.

			Total Solution Time		Time to Optimum	
m	n	T	SDA	BB	SDA	BB
10	30	5	2.94	4.50	2.07	3.84
10	30	5	4.48	11.12	3.10	8.37
12	24	3	2.46	10.11	.89	9.40
12	24	3	3.04	5.87	1.54	5.38
30	6	3	1.55	3.74	1.00	2.45
30	12	3	4.14	18.36	2.82	17.91
15	15	3	1.88	4.32	1.08	4.32
15	15	3	2.10	4.17	1.00	4.16
18	18	3	1.32	3.06	.74	2.80
20	20	4	1.51	5.27	1.08	2.61
20	20	5	1.74	3.86	1.24	3.29
24	24	4	2.00	4.03	1.82	2.68

Table 2: Results for Some Small Test Problems

The decomposition algorithm also performs well on most moderate-sized binary test problems. As demonstrated by the results in Table 3, the SDA can often solve a problem with 40 or 50 variables in a relatively small fraction of the time used by BB. This behavior holds over an extremely wide range of test problems.

Ordinarily the decomposition technique will require one-third to one-tenth as many seconds as the non-decomposition method, although even greater disparities can occur. Occasionally, however, the decomposition approach will not fare so well; there are certain types of problems which are more difficult for the SDA than for BB. An example of such a problem is given in the last line of Table 3. One unfavorable characteristic of this problem is that the right-hand sides of the constraints are quite large, so that the look-ahead prices are too small in magnitude to effectively guide the subproblem search. In this particular case, the search directions chosen due to penalty calculations are rather poor. Fortunately this situation does not occur often, and even under these adverse circumstances the solution times for the SDA are competitive with those of BB.

Due to limitations on computer resources, the testing of problems with more than 60 variables was not as extensive as the examination of more moderate-sized problems. A selection of the results on larger or more difficult problems is given in Table 4. For some of the sample problems presented there, BB exceeded a reasonable time limit* just trying to find an optimal solution, while the SDA was able to discover and verify an optimal solution in approximately one-fifth the time.

*Computations were halted after 100,000 simplex iterations for BB and 1,000,000 simplex iterations for the SDA.

			Total Solution Time		Time to Optimum	
m	n	T	SDA	BB	SDA	BB
18	30	3	19.9	77.0	10.1	31.6
18	42	6	7.6	119.1	3.5	108.1
20	40	5	39.1	126.1	34.8	108.4
20	50	5	43.2	139.4	27.1	55.5
24	32	4	15.1	112.2	12.7	32.0
42	30	6	5.5	112.6	2.8	88.9
50	30	5	5.7	100.6	5.3	68.3
20	52	4	114.0	592.3	58.7	47.1
30	40	5	16.9	293.3	5.0	13.2
30	30	5	5.5	56.4	4.6	47.4
30	30	3	29.2	86.8	21.7	14.9
36	36	6	2.8	26.5	2.4	22.8
40	40	4	111.7	813.5	5.3	68.3
20	40	4	51.7	34.0	31.1	26.0

Table 3: Results for Some Moderate-Sized Problems

			Total Solution Time		Time to Optimum	
m	n	T	SDA	BB	SDA	BB
28	56	7	216	952	112	559
20	60	4	552	2142	261	897
20	60	4	48	101	37	91
30	60	5	110	4335	9	1735
30	60	5	222	>4737	72	1427
50	50	5	923	6149	570	251
50	50	5	1182	>5700	900	>5700
54	54	9	393	1210	339	125
50	100	5	133	928	39	712
50	100	5	355	>9495	290	397
50	100	5	931	>7536	689	>7536
20	100	10	75	1	71	1

Table 4: Results for Some Large Problems

The final line in Table 4 shows the test results of a large problem having a binary constraint matrix. The solution to the LP relaxation of this problem was naturally integral, as is sometimes the case with binary data. This property enabled BB to solve the problem without taking a single branch. On the other hand, the SDA could not take advantage of this fortuitous linear programming solution, because such information is unavailable after the problem has been decomposed. This example clearly illustrates one of the limitations of the SDA: it should not be used on a problem with a unimodular constraint matrix.

3. The Influence of Various Problem Parameters

In the preceding section, empirical evidence was given to demonstrate the effectiveness of the decomposition method. That effectiveness can be enhanced or diminished by changing the input parameters which determine the characteristics of a randomly generated test problem. The effects of problem size and structure on solution times will be examined in this section.

Of principal concern for any integer programming algorithm is the growth rate of solution times as the number of integer variables increases. As for all general purpose integer programming methods developed to date (see Chapter 1), an apparently exponential rate of growth has been observed for the branch-and-bound code BB. Since BB is incorporated as an important subroutine in the SDA, the latter must share the exponential growth characteristic of the former. Figure 10 depicts the growth rate of the SDA and BB on a test problem having 30 rows and 5 periods, as the number of binary variables increases from 20 to 60. The coefficients of the original variables remain the same as new variables are appended to the problem.

To further illustrate the effects of increasing the number of columns, Table 5 presents computational results from another sample problem. Throughout extensive testing, the SDA has maintained a consistent advantage over BB, whether the number of variables is large or small.

One parameter which has a strong influence on the relative solution times of the SDA and BB is the number of rows. As the number of constraints grows, the look-ahead prices become a more reliable criterion for guiding the branch-and-bound search, and the fathoming bound $\max c(\lambda,k)$

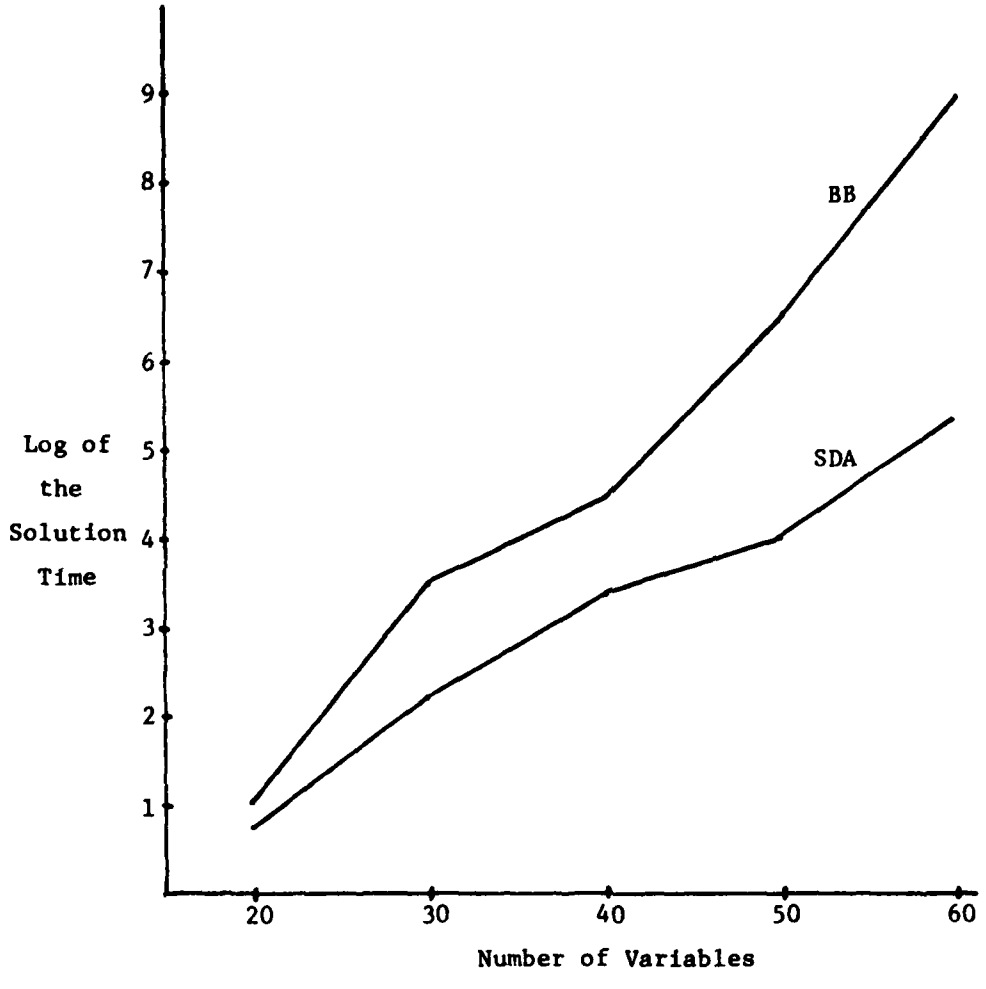


Figure 10: Exponential Growth of Solution Time

			Total Solution Time		Time to Optimum	
m	n	T	SDA	BB	SDA	BB
20	20	4	1	5	1	3
20	28	4	4	29	2	27
20	36	4	9	41	7	35
20	44	4	63	200	59	195
20	52	4	114	592	59	47
20	60	4	552	2142	261	897

Table 5: Effect of Increasing the Number of Variables

is strengthened. On the other hand, the linear programming subroutines must work harder to solve the larger LP relaxations at each node. This latter effect causes more difficulty for BB than for the SDA, since decomposition lessens the impact of the increase in the number of rows. An increase in the number of constraints therefore is more likely to benefit the SDA. A typical example of the results obtained when extra constraints are appended to a problem is shown in Table 6.

			Total Solution Time		Time to Optimum	
m	n	T	SDA	BB	SDA	BB
10	30	5	3	5	2	4
20	30	5	9	32	7	11
30	30	5	14	192	6	154
40	30	5	31	353	9	25
50	30	5	32	469	22	103

Table 6: Effect of Increasing the Number of Rows

As the number of periods increases, so does the effectiveness of the decomposition approach, at least for a while. A sequence of randomly generated problems with similar characteristics, except for the number of periods*, was tested to demonstrate the consequences of varying T. The outcomes for these sample problems are exhibited in Table 7. (When the number of periods did not divide 40, the rows and columns were distributed as evenly as possible, so that no period had more than one extra constraint or variable.)

			Total Solution Time		Time to Optimum	
m	n	T	SDA	BB	SDA	BB
40	40	4	81	473	38	189
40	40	5	81	1050	64	381
40	40	6	83	1397	54	1092
40	40	7	138	1670	100	1374
40	40	8	21	270	5	75
40	40	9	44	368	6	95
40	40	10	17	221	14	162

Table 7: Effect of Increasing the Number of Periods

*Unlike the preceding 3 sample problems, the change in parameter T does also imply a change in all the problem coefficients. This is an unfortunate result of the structure of the data generation program.

The tightness of the problem constraints can have a strong influence on the comparative solution times of BB and the SDA. Large values for the right-hand sides can produce enough slack in the problem (R_k) to make the look-ahead prices $P_k(j)$ ineffective (see Chapter 3). For this reason, increasing the tightness ratio τ defined in Section 1 of this chapter leads to a decrease in the efficiency of the decomposition approach. This fact is illustrated in Table 8 on a problem with 20 rows, 30 columns, and 5 periods. All constraint and objective function coefficients are held fixed as τ increases from 2 to 10.

	Total Solution Time		Time to Optimum	
τ	SDA	BB	SDA	BB
2	2	12	1	5
4	7	34	2	4
6	9	50	3	41
8	15	53	9	39
10	2	5	2	3

Table 8: Effect of Decreasing Constraint Tightness

The decomposition approach appears to be most advantageous when the diagonal blocks are moderately dense. This effect, which is exemplified in Table 9, is largely attributable to an increase in the efficiency of the branch-and-bound algorithm BB as the sparsity of the whole constraint matrix increases. Thus the SDA seems to improve in comparison

as the density increases from low to moderate values. The sample problem shown in Table 9 has 40 rows, 40 columns, and 4 periods. The density of this problem was decreased by changing the appropriate proportion of (randomly chosen) coefficients to zero.

	Total Solution Time		Time to Optimum	
Density	SDA	BB	SDA	BB
.2	6	16	4	12
.3	9	49	8	35
.4	18	100	18	16
.5	16	458	13	238
.6	26	620	4	170
.7	40	407	35	94
.8	44	434	37	101
.9	48	206	41	41
1.0	82	140	74	28

Table 9: Effect of Increasing Diagonal Block Density

All of the results given thus far have been for binary, i.e., (0,1) integer programming test problems. One explanation for this is that problems with wider ranges for the variable bounds can take quite long to solve. However it is also true that the SDA does not perform as well on such problems. This is due, in part, to the fact that the look-ahead prices are a measurement of the relative effects of increasing

AD-A089 542 STANFORD UNIV CA DEPT OF OPERATIONS RESEARCH F/G 12/2
THE STAIRCASE AND RELATED STRUCTURES IN INTEGER PROGRAMMING.(U)
JUN 80 L J POLLENZ N00014-76-C-0418
UNCLASSIFIED TR-94 NL

each variable from 0 to 1, and are somewhat less valid over a wider range (see Chapter 3 for further explanation). Judgement on the merits of the decomposition method should be withheld in this case, pending the implementation of pseudocosts as a replacement for the look-ahead prices. Table 10 gives a small sample of results for the SDA on some non-binary problems.

				Total Solution Time		Time to Optimum	
Range	m	n	T	SDA	BB	SDA	BB
(0,2)	30	30	3	56	121	36	88
(0,2)	20	20	4	5	13	4	8
(0,2)	30	30	6	25	9	20	4
(0,3)	30	30	6	38	38	29	20
(0,5)	30	30	6	444	92	382	41

Table 10: Results for Some Non-Binary Problems

To conclude this investigation into the effects of varying test problem characteristics, Tables 11 and 12 show the impact of uniformly increasing the problem dimensions. For a multitime period model, this would correspond to expanding the time horizon of the study. For both of the sample problems shown below, the advantage of the SDA increases with overall problem size. Test results of this kind are a strong indication that decomposition would prove beneficial even for problems larger than those examined in this computational study.

			Total Solution Time		Time to Optimum	
m	n	T	SDA	BB	SDA	BB
18	18	3	1	1	1	1
24	24	4	2	4	2	3
30	30	5	2	10	2	4
36	36	6	7	49	5	34
42	42	7	14	59	11	11
48	48	8	24	128	18	17
54	54	9	49	537	37	250

Table 11: Effect of Increasing Problem Dimensions

			Total Solution Time		Time to Optimum	
m	n	T	SDA	BB	SDA	BB
12	24	3	3	6	2	5
16	32	4	47	72	42	63
20	40	5	39	126	35	108
24	48	6	139	714	117	502
28	56	7	433	2032	370	716

Table 12: Effect of Increasing Problem Dimensions

4. Suboptimal Solution Times

As evidenced by the computational results presented in the preceding two sections, the staircase decomposition algorithm not only terminates faster than the branch-and-bound method, it also usually discovers (without yet verifying) an optimal solution more rapidly. Quite often the difference in time to reach an optimal solution can be an order of magnitude or more. This fact is remarkable because the search directions produced by BB should be superior to those of the SDA (due to the extra information available from the full simplex tableau).

Although the branch-and-bound method will occasionally arrive at an optimum solution faster than the decomposition technique, the SDA normally finds a good feasible point more quickly. For the vast majority of problems the SDA will swiftly find an integer point within 5 or 10% of the optimum objective value. A variety of sample results are given in Table 13 to demonstrate the speed of the SDA in reaching good feasible solutions. Note the initial dominance of the SDA even on problems for which the total execution times of BB and the SDA are comparable. The final line of Table 13 shows that the SDA does not really perform so badly under this criterion on the problem with binary data shown previously in Table 4.

The extremely rapid arrival of the SDA at near-optimal solutions implies that it would be an excellent choice if computer resources are limited. It would also serve well as an heuristic for generating a good initial feasible solution to use as input to a method like Hillier's bound-and-scan algorithm (see Chapter 1).

			Total Solution Time		Time to 95% of Optimum	
m	n	T	SDA	BB	SDA	BB
20	20	2	16	14	2	4
20	40	5	94	120	2	62
20	60	5	1120	1521	30	507
30	50	5	58	886	3	73
30	60	5	110	4335	5	606
30	60	5	222	>4737	8	76
36	30	3	35	66	5	24
40	40	4	100	317	4	64
40	40	4	102	1725	10	728
50	100	5	133	928	39	712
50	100	5	1337	>7759	14	295
20	100	10	75	1	5	1

Table 13: Rapid Arrival at Near-Optimum Solutions

On some of the more difficult problems tested (with 60 or fewer variables), BB required 30 CPU seconds to reach any feasible solution at all. For all problems of that size, the SDA first encounters a feasible solution within 1 to 6 seconds, regardless of problem difficulty. Thus it seems that the decomposition approach is superior to the branch-and-bound method in the early stages of execution, as well as terminating more quickly for most problems.

CHAPTER 7: CONCLUSIONS

1. Summary

Staircase structured integer programming models arise in a number of applications, including multiplant production allocation problems and multitime period production and inventory problems. Conventional pure or mixed integer programming methods for solving these problems may be quite time-consuming, particularly if the models are large. The staircase decomposition algorithm described in Chapters 2 and 3 should be a better solution technique because it takes advantage of the inherent structure of the problem.

By decomposing the original problem into small subproblems (one for each diagonal block of the constraint matrix), the SDA can execute each branch very efficiently. Compared to a standard branch-and-bound algorithm, it uses less CPU time and a smaller portion of core. Guided by look-ahead prices and Tomlin's improved penalties, this algorithm quickly arrives at an optimal solution. The fathoming criteria derived from the bounds on future objective value developed in Chapter 4 allow rapid verification of optimality, so that total execution time is relatively small.

Computational results, on a wide variety of randomly generated test problems, have been very encouraging. These empirical results involve the comparison of the staircase decomposition method with a standard branch-and-bound algorithm (BB) applied to the undecomposed problem. This comparison is particularly valid because BB also serves

as the subproblem search procedure for the SDA. Thus any flaws or inefficiencies which might be present in the computer code BB would also affect the performance of the SDA.

For the majority of the moderate-sized problems tested, the decomposition technique discovered and verified an optimal solution 3 to 10 times faster than the standard branch-and-bound algorithm. These impressive results held true for small and large problems as well, and under varying problem dimensions, density, and tightness of constraints. The effectiveness of the SDA increased significantly as the constraints were made tighter or more numerous, and decreased as the bounds on the variables were widened beyond [0,1].

In addition to its advantage in total execution time, the SDA consistently found near-optimum integer solutions much faster than the branch-and-bound method applied to the undecomposed problem. For this reason, and also because of its smaller core requirements, the decomposition method could prove even more valuable if computer resources are limited.

As explained in detail in Chapter 5, the staircase decomposition algorithm can easily be modified to solve higher order staircase problems, nonlinear integer programming problems (with the nonlinearities concentrated in the offdiagonal blocks), and mixed integer linear programs. The ability to solve mixed integer programs is of particular importance, since the formulations of many real world applications of the staircase model contain both integer and continuous variables.

2. Directions for Future Research

One computational modification of the SDA which should be explored is the substitution of pseudocosts (see Forrest et al. [1974] and Gauthier and Ribiere [1977]) for look-ahead prices and penalties. After enough data are obtained to provide a good basis for pseudocosts, they provide a more accurate estimate of the actual degradation in objective value per unit change of each integer variable than penalties. Moreover, pseudocosts are not calculated in a static manner, as are the look-ahead prices; they are updated as the branch-and-bound search progresses. Therefore, a branch-and-bound subproblem search guided by pseudocosts should be more efficient than the more rigid search strategy which results from the use of look-ahead prices.

For certain problems, such as higher order staircase problems, loosely constrained problems, and non-binary problems, the look-ahead prices are less likely to be effective in guiding the subproblem search procedure towards a global optimum. The algorithmic improvement due to the replacement of look-ahead prices by pseudocosts should be especially significant for these types of problems.

In addition to the use of pseudocosts, alternative search strategies might enhance the performance of the SDA. The last-in-first-out (LIFO) search strategy of Kochman's branch-and-bound algorithm is the best in terms of minimal programming complexity, but other strategies might lead to the optimum more quickly. Under one alternative strategy, the most promising node remaining on the node list is investigated next. For this approach to improve on the performance of the LIFO strategy, the extra bookkeeping work must be compensated for by the faster arrival at

an optimal solution. Even more appealing is the idea of fathoming certain nodes temporarily when that particular branch does not appear to be leading towards a significant improvement in objective value. Two possible less stringent fathoming criteria are given by (32) and (33).

$$(32) \qquad \sum_{t=1}^{k-1} c_t x_t + (1 - \lambda) z_k + \text{maxc}(\lambda, k) + \min_j \{P_k(j)\} \leq c\bar{x}$$

$$(33) \qquad \sum_{t=1}^{k-1} c_t x_t + (1 - \lambda) z_k + \text{maxc}(\lambda, k) \leq (1 + \varepsilon) c\bar{x}$$

In (32), the price $P_k(j)$ could be replaced by a pseudocost for rounding off variable j. Using this fathoming criterion, a (perhaps suboptimal) solution will be discovered by the decomposition algorithm. At the end of the search procedure, the SDA could return to these temporarily fathomed nodes to check for further improvement in objective value.

If criterion (33) is used for fathoming, an ε-optimal solution will be found by the SDA. For some purposes, this solution will suffice; if not, the nodes fathomed early (because $\varepsilon > 0$) can be explored at the end of the search procedure. Often many of these nodes will be fathomed (with $\varepsilon = 0$) immediately because of the improvement in objective value from the time at which these nodes were temporarily fathomed. Compared to the LIFO search strategy, fewer nodes will have to be explored, and a near optimum solution should be found more quickly.

Another factor which might influence the performance of the SDA is the degree of decomposition of a staircase problem. For example, a multitime period model with 12 periods can be expressed as a staircase problem with 2, 3, 4, or 6 periods simply by aggregating adjacent periods. The higher the degree of decomposition, the faster branches can be taken, but also the less information is available to guide the subproblem search. For a problem with a large number of time periods, it is therefore likely that the optimal level of decomposition would be somewhat less than the maximum degree possible. Further computational results on this subject are required to pinpoint the optimal degree of decomposition for a large problem.

As discussed in Chapter 4, the "optimal" choice of $\{\lambda_i, i = 1,\ldots,r\}$ for use in fathoming is not obvious. It would be useful to investigate the tradeoff between the increase in fathoming power due to enlarging the number of bounds generated and the extra computational work needed to calculate and test these bounds. In addition, the speculation that the best choice for the values λ_i, for $i = 1, \ldots, r$, is to space them evenly over the interval $[0,(r - 1)/r]$ (i.e., $\lambda_i = (i - 1)/r$) should be subjected to computational testing.

Perhaps the most important unresolved question concerns the effectiveness of the staircase decomposition algorithm on real world problems. The empirical evidence given in Chapter 6 is very promising, but actual problems may not behave in the same manner as randomly generated ones. Many of the applications of multitime period models are mixed integer, often with strictly local continuous variables. These have not yet been

tested. Also, most real world problems discount costs over time, which should benefit the SDA (because it automatically gives priority in the branching strategy to variables in the early periods). The indications are that the decomposition approach will be an efficient solution technique for real staircase problems, but this remains to be proven conclusively.

REFERENCES

Abrahamson, Philip, private communication (1980).

Aho, Alfred V., John E. Hopcroft and Jeffrey D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley Publishing Company, Reading, Massachusetts (1974).

Balas, Egon, "Intersection Cuts from Disjunctive Constraints." Management Science Research Report 330, Carnegie-Mellon University (1974).

----- and R.G. Jeroslow, "Strengthening Cuts for Mixed Integer Programs." Management Science Research Report 359, Carnegie-Mellon University (1975).

Benders, J.F., "Partitioning Procedures for Solving Mixed Variables Programming Problems." Numerische Mathematik 4 (1962), 238-252.

Benichou, M., J.M. Gauthier, G. Hentges, and G. Ribiere, "The Efficient Solution of Large-Scale Linear Programming Problems -- Some Algorithmic Techniques and Computational Results." Mathematical Programming 13 (1977), 280-322.

Cooper, L. and M.W., "All-Integer Programming -- A New Approach Via Dynamic Programming." Naval Research Logistics Quarterly 25 (1978), 415-429.

Dantzig, George B., Linear Programming and Extensions, Princeton University Press, Princeton, New Jersey (1963).

-----,"Solving Staircase Linear Programs by a Nested Block-Angular Method." Technical Report 73-1, Department of Operations Research, Stanford University (1973).

-----, and Philip Wolfe, "Decomposition Principle for Linear Programs." Operations Research 8 (1960), 101-111.

Driebeek, Norman J., Applied Linear Programming, Addison-Wesley Publishing Company, Reading, Massachusetts (1969), Chapter 5.

Denardo, Eric V. and Bennett L. Fox, "Shortest-Route Methods: 2. Group Knapsacks, Expanded Networks, and Branch-and-Bound." Operations Research 27 (1979), 548-566.

Faaland, Bruce H. and Frederick S. Hillier, "The Accelerated Bound-and-Scan Algorithm for Integer Programming." Operations Research 23 (1975), 406-425.

-----, and Frederick S. Hillier, "Interior Path Methods for Heuristic Integer Programming Procedures." Operations Research 27 (1979), 1069-1087.

Forrest, J.J.H., J.P.H. Hirst and J.A. Tomlin, "Practical Solution of Large Mixed Integer Programming Problems with UMPIRE." Management Science 20 (1974), 736-773.

Fourer, Robert F., "Solving Staircase Linear Programs by the Simplex Method." Ph.D. Dissertation, Department of Operations Research, Stanford University (1980).

Gacs, Peter and Laszlo Lovasz, "Khachian's Algorithm for Linear Programming." Report 79-750, Department of Computer Science, Stanford University (1979).

Garfinkel, R.S. and G.L. Nemhauser, Integer Programming, Wiley & Sons, New York (1972).

-----, and G.L. Nemhauser, "A Survey of Integer Programming Emphasizing Computation and Relations Among Models." Mathematical Programming, T.C. Hu and Stephen M. Robinson, editors, Academic Press, New York (1973), 77-155.

Gauthier, J.M. and Ribiere G., "Experiments in Mixed-Integer Linear Programming Using Pseudo-costs." Mathematical Programming 12 (1977), 26-47.

Geoffrion, A.M., "An Improved Implicit Enumeration Approach to Integer Programming." Operations Research 17 (1969), 437-454.

-----, "Lagrangean Relaxation for Integer Programming." Mathematical Programming Study 2 (1974), 82-114.

-----, "A Guided Tour of Recent Practical Advances in Integer Linear Programming." OMEGA 4 (1976), 49-57.

-----, and G.W. Graves, "Multicommodity Distribution System Design by Benders Decomposition." Management Science 20 (1974), 822-844.

-----, and R.E. Marsten, "Integer Programming Algorithms: A Framework and State-of-the-Art Survey." Management Science 18 (1972), 465-491.

-----, and R. Nauss, "Parametric and Postoptimality Analysis in Integer Linear Programming." Management Science 23 (1977), 453-466.

Glassey, C. Roger, "Dynamic Linear Programs for Production Scheduling." Operations Research 19 (1971), 45-56.

-----, "Nested Decomposition and Multi-Stage Linear Programs." Management Science 20 (1973), 282-292.

Glover, Fred, "Surrogate Constraints." Operations Research 16 (1968), 741-749.

-----, "Parametric Branch and Bound." OMEGA 6 (1978), 145-152.

Gomory, R.E., "An Algorithm for Integer Solutions to Linear Programs." Recent Advances in Mathematical Programming, R. Graves and P. Wolfe, editors, McGraw-Hill, New York (1963), 269-302.

-----, "On the Relation Between Integer and Noninteger Solutions to Linear Programs." Proceedings of the National Academy of Science 53 (1965), 260-265.

Gorry, G. and J. Shapiro, "An Adaptive Group Theoretic Algorithm for Integer Programming Problems." Management Science 17 (1971), 285-306.

Heesterman, A.R.G. and J. Sandee, "Special Simplex Algorithm for Linked Problems." Management Science 11 (1965), 420-428.

Hillier, Frederick S., "A Bound-and-Scan Algorithm for Integer Linear Programming with General Variables." Operations Research 17 (1969), 638-679.

Ho, James K., "Optimal Design of Multi-Stage Structures: A Nested Decomposition Approach." Computers and Structures 5 (1975), 249-255.

-----, "Nested Decomposition of a Dynamic Energy Model." Management Science 23 (1977), 1022-1026.

-----, and Alan S. Manne, "Nested Decomposition for Dynamic Models." Mathematical Programming 6 (1974), 121-140.

Holm, Soren and Dieter Klein, "Discrete Right Hand Side Parametrization of Linear Integer Programs." European Journal of Operations Research 2 (1978), 50-53.

Ibaraki, Toshihide, "On the Computational Efficiency of Branch-and-Bound Algorithms." Journal of the Operations Research Society of Japan 20 (1977), 16-35.

Kaneko, I., "On the Unboundedness of the Set of Integral Points in a Polyhedral Region." Technical Report SOL 74-12, Systems Optimization Laboratory, Department of Operations Research, Stanford University (1974).

Karp, Richard M., "Probabilistic Analysis of Partitioning Algorithms for the Traveling Salesman Problem in the Plane." Mathematics of Operations Research 2 (1977), 209-224.

Kochman, Gary A., "Decomposition in Integer Programming." Technical Report 66, Department of Operations Research, Stanford University (1976a).

-----, "Computer Programs for Decomposition in Integer Programming." Technical Report 71, Department of Operations Research, Stanford University, (1976b).

Kochman, Gary A., and Lynne J. Pollenz, "A Decomposition Algorithm for the Uncapacitated Plant Location Problem." Presented at the ORSA Conference in New York (May 1978).

Land, A.H. and A.G. Doig, "An Automatic Method of Solving Discrete Programming Problems." Econometrica 28 (1960), 497-520.

Lasdon, Leon S., Optimization Theory for Large Systems, The MacMillan Company, New York (1970), Chapter 2.

Lawler, E.L. and D.E. Wood, "Branch-and-Bound Methods: A Survey." Operations Research 14 (1966), 699-719.

Lenstra, J.K. and A.H.G. Rinnooy Kan, "On the Expected Performance of Branch-and-Bound Algorithms." Operations Research 26 (1978), 347-349.

Luenberger, David G., Introduction to Dynamic Systems: Theory, Models, and Applications, Wiley & Sons, New York (1979).

Madsen, Oli B.G., "Solution of Linear Programming Problems with Staircase Structure." Research Report 26, The Institute of Mathematical Statistics and Operations Research, Lyngby, Denmark (1977).

Manne, Alan S., "Sufficient Conditions for Optimality in an Infinite Horizon Development Plan." Econometrica 38 (1970), 18-38.

Orchard-Hays, William, Advanced Linear Programming Computer Techniques, McGraw-Hill, New York (1968).

Perold, Andre F. and George B. Dantzig, "A Basis Factorization Method for Block Triangular Linear Programs." Technical Report SOL 78-7, Systems Optimization Laboratory, Department of Operations Research, Stanford University (1978).

Reardon, Kevin J., "A Decomposition Method for the Solution of Dual-Angular Integer Programs." Technical Report 51, Department of Operations Research, Stanford University (1974).

Rosen, J.B., "Primal Partition Programming for Block Diagonal Matrices." Numerische Mathematik 6 (1964), 250-260.

Saigal, Romesh, "Block-Triangularization of Multi-Stage Linear Programs." Report ORC 66-9, Operations Research Center, University of California, Berkeley (1966).

Schrage, Linus, "Using Decomposition in Integer Programming." Naval Research Logistics Quarterly 19 (1972), 435-447.

Shapiro, J.F., "Dynamic Programming Algorithms for the Integer Programming Problem I: The Integer Programming Problem Viewed as a Knapsack Type Problem." Operations Research 16 (1968a), 103-121.

Shapiro, J.F., "Group Theoretic Algorithms for the Integer Programming Problem II: Extension to a General Algorithm." Operations Research 16 (1968b), 928-947.

Tomlin, John A., "An Improved Branch-and-Bound Method for Integer Programming." Operations Research 19 (1971), 1070-1075.

Trauth, C.A. and R.E. Woolsey, "Integer Linear Programming: A Study in Computational Efficiency." Management Science 15 (1969), 481-493.

Weil, R.L. and P.C. Kettler, "Rearranging Matrices to Block-Angular Form for Decomposition (and Other) Algorithms." Management Science 18 (1971), 98-108.

Wollmer, Richard D., "A Substitute Inverse for the Basis of a Staircase Structure Linear Program." Mathematics of Operations Research 2 (1977), 230-239.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 94	2. GOVT ACCESSION NO. AD-A089 542	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) THE STAIRCASE AND RELATED STRUCTURES IN INTEGER PROGRAMMING		5. TYPE OF REPORT & PERIOD COVERED TECHNICAL REPORT 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) LYNNE POLLENZ		8. CONTRACT OR GRANT NUMBER(s) N00014-76-C-0418
9. PERFORMING ORGANIZATION NAME AND ADDRESS OPERATIONS RESEARCH PROGRAM ONR Department of Operations Research Stanford University, Stanford, California		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR-047-061
11. CONTROLLING OFFICE NAME AND ADDRESS OFFICE OF NAVAL RESEARCH OPERATIONS RESEARCH PROGRAM CODE 434 ARLINGTON, VA. 22217		12. REPORT DATE JUNE 1980 13. NUMBER OF PAGES 108
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Also issued as Technical Report No. 80-13, Dept. of Operations Research Stanford University, under National Science Foundation Grant MCS76-81259

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

INTEGER PROGRAMMING

DECOMPOSITION

STAIRCASE

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

SEE OTHER SIDE

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TECHNICAL REPORT NO. 94 Author: Lynne Pollenz

Staircase structured integer programming problems arise in a wide variety of practical applications, including multitime period production and inventory problems and multisector economic planning models. In order to exploit the special structure inherent in a staircase model, a decomposition algorithm has been developed. This algorithm can be modified to solve lower block triangular problems and staircase mixed integer linear programs. Computational results, including testing against a standard integer programming code, have been very encouraging.

. . . . . .